UNISYS

A Series
DMSII
Technical Overview

Release Mark 3.8

Priced Item

October 1989
Distribution Code SE
Printed in U S America
5044191.380

UNISYS

A Series
DMSII
Technical Overview

Copyright © 1989 Unisys Corporation
All Rights Reserved
Unisys is a trademark of Unisys Corporation

Release Mark 3.8

October 1989
Distribution Code SE
Printed in U S America

Priced Item
5044191.380

The names, places, and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual living or otherwise, or that of any group or association, is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special, or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication may be forwarded using the Product Information card at the back of the manual, or may be addressed directly to Unisys, Technical Publications, 460 Sierra Madre Villa, Pasadena, CA 91109.

Page Status

Page	Issue
iii	Original
iv	Blank
v through ix	Original
x	Blank
xi through xvii	Original
xviii	Blank
1-1 through 1-7	Original
1-8	Blank
2-1 through 2-39	Original
2-40	Blank
3-1 through 3-7	Original
3-8	Blank
4-1 through 4-17	Original
4-18	Blank
5-1 through 5-2	Original
6-1 through 6-12	Original
7-1 through 7-3	Original
7-4	Blank
8-1 through 8-4	Original
Glossary-1 through 11	Original
Glossary-12	Blank
Bibliography-1 through 2	Original
Index-1 through Index-9	Original
Index-10	Blank

About This Overview

Purpose

The *A Series DMSII Technical Overview* provides an overview of the Data Management System II (DMSII) environment, which includes DMSII databases and software. Because it is intended as an introduction to DMSII, you should read the body of this overview before you read the other DMSII manuals.

Scope

This overview explains DMSII database structures and discusses how those structures interact with the software in the DMSII environment. Secondly, it describes generally how to create, change, restore, access, and maintain a DMSII database.

Audience

The audience for this overview includes:

- Application and system programmers who need to understand DMSII to create user programs that use a DMSII database
- People who have worked on other types of database management systems and now need to understand how DMSII works
- People who are responsible for maintaining DMSII databases

Prerequisites

Reading the body of this overview requires some background knowledge of programming.

How to Use This Overview

First read this overview sequentially. You can use parts of it later as a reference, or when you need a quick refresher of DMSII.

The examples of this overview use Data and Structure Definition Language (DASDL) and COBOL74 syntax. For more information on DASDL, see the *A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual*. For more information on COBOL74, see the *A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces*.

This overview uses the word *access* in two ways: with a capital *A* and a small *a*. *Access* with a capital *A* refers to a database structure. Without an initial capital, *access* means to use or retrieve something.

Organization

In addition to the following sections, this overview includes a glossary, a bibliography, and an index.

Section 1. Introducing the Data Management System II (DMSII)

This section introduces the basic functions and framework of DMSII and how they interact. This framework includes DASDL, the description file, the Accessroutines, the control file, and the DMSUPPORT library.

Section 2. Understanding DMSII Database Concepts

This section describes the parts of a DMSII database, including data sets, data items, sets, subsets, and links. It also shows how you can represent the parts of a business in a database.

Section 3. Ensuring DMSII Database Security

This section describes how remaps, logical databases, and guard files provide security for DMSII databases.

Section 4. Creating a DMSII Database

This section discusses how to collect information and use it to implement a DMSII database.

Section 5. Changing a DMSII Database

This section describes update, reorganization, and modeling, which are processes that change the format or data of a database.

Section 6. Restoring a DMSII Database

This section describes how to recover the database in case of a system failure. It discusses backing up, auditing, and recovering the database.

Section 7. Accessing a DMSII Database

This section describes how to retrieve and manipulate data in the database using Extended Retrieval with Graphic Output (ERGO) or application programs.

Section 8. Maintaining a DMSII Database

This section discusses database administration functions using the Operations Control Manager (OCM), DBANALYZER, DBCERTIFICATION, DMMONITOR, DMUTILITY, LOADDUMP, and VISIBLE DBS.

Related Product Information

***A Series ALGOL Programming Reference Manual, Volume 2: Product Interfaces* (form 1170099)**

This manual describes extensions to the Extended ALGOL language designed to allow application programs to interface with the Advanced Data Dictionary System (ADDS), the Communications Management System (COMS), DMSII, the Screen Design Facility Plus (SDF Plus), or the Semantic Information Manager (SIM). This manual is written for programmers who are familiar with Extended ALGOL programming language concepts and terms.

***A Series CANDE Operations Reference Manual* (form 1169869)**

This manual describes how CANDE operates to allow generalized file preparation and updating in an interactive, terminal-oriented environment. This manual is written for a wide range of computer users who work with text and program files.

***A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces* (form 1180619)**

This manual describes the extensions to the standard COBOL74 language. These extensions are designed to allow application programs to interface with ADDS, COMS, DMSII, the DMSII Transaction Processing System (TPS), the Screen Design Facility (SDF), SDF Plus, and SIM products. This manual is written for programmers who are familiar with COBOL74 programming language concepts and terms.

***A Series Communications Management System (COMS) Operations Guide* (form 1154523)**

This guide explains how to perform terminal-based COMS functional tasks and serves as a reference to COMS commands. Syntax diagrams of COMS commands are provided with explanations and examples of how the commands can be used. Examples are provided for most command contexts. This guide is written for terminal operators and computer operators.

***A Series DMSII Application Program Interfaces Programming Guide* (form 5044225). Formerly *A Series DMSII User Language Interface Programming Guide* (form 1180536).**

This guide explains how to write effective and efficient application programs that access and manipulate a DMSII database using either the DMSII interpretive interface or the DMSII language extensions. This guide is written for application programmers and database administrators who are already familiar with the basic concepts of DMSII.

***A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual* (form 1163805)**

This manual provides instructions for defining and maintaining a DMSII database using DASDL. This manual is written for database administrators and staff.

***A Series DMSII Interpretive Interface Programming Reference Manual* (form 1163813)**

This manual details the DMSII interpretive interface funtions, including generating the DMINTERPRETER library, using entry points, and coding application programs. This manual is written for application programmers and database administrators who are already knowledgeable in the basic concepts of DMSII.

***A Series DMSII Utilities Operations Guide* (form 1163839)**

This guide describes how to maintain relationships between data elements in a DMSII database. This guide is written for database administrators and programmers who are responsible for database integrity and recovery.

***A Series Extended Retrieval with Graphic Output (ERGO) Operations Guide* (form 1164027)**

This guide explains how to use ERGO to access DMSII databases. This guide is written for managers, programmers, analysts, and support personnel involved in preparing reports from information in DMSII databases.

***A Series Generalized Message Control System (GEMCOS) Operations Guide* (form 1185329)**

This guide explains how to use, manage, and maintain A Series GEMCOS message control systems (MCSs). This guide is written for application programmers and system administrators.

***A Series InfoExec™ Administration Guide* (form 1195070)**

This guide describes the administration concepts and procedures of the InfoExec data management system. This guide is written for individuals responsible for administration of a database or data dictionary.

***A Series InfoExec Operations Guide* (form 1195120)**

This guide describes InfoExec operations including the use of system commands to control the data dictionary and the use of InfoExec screens to perform database operations such as recovery, initialization, and reorganization. This guide is written for those who will install and operate InfoExec products and who are familiar with DMSII and have read the *A Series InfoExec Functional Overview* and the *A Series InfoExec Semantic Information Manager (SIM) Technical Overview*.

InfoExec is a trademark of Unisys Corporation.

***A Series Pascal Programming Reference Manual, Volume 2: Product Interfaces* (form 1170107)**

This manual describes the extensions to the Pascal language designed to allow application programs to interface with ADDS, COMS, SDF Plus, or SIM. This manual is written for applications programmers who are familiar with Pascal programming language concepts and terms.

***A Series Report Program Generator (RPG) Programming Reference Manual, Volume 2: Product Interfaces* (form 1170115)**

This manual describes extensions to the RPG language designed to allow application programs to interface with DMSII or COMS. This manual is written for applications programmers who are familiar with RPG programming language concepts and terms.

***A Series Security Features Operations and Programming Guide* (form 1195203)**

This guide describes the security features available to users and provides instructions on their use. This guide is written for users who are responsible for maintaining the security of their individual programs and data.

***A Series System Commands Operations Reference Manual* (form 1169612)**

This manual gives a complete description of the system commands used to control system resources and work flow. This manual is written for systems operators and administrators.

Contents

About This Overview v

Section 1. Introducing the Data Management System II (DMSII)

Functions of the DMSII Environment 1–1
DMSII Framework 1–2
DASDL 1–3
Description File 1–3
Accessroutines 1–3
Control File 1–4
DMSUPPORT Library 1–4
How the DMSII Framework and Software Interact 1–4
Interactions in Creating a Database 1–5
Interactions in Accessing a Database 1–6

Section 2. Understanding DMSII Database Concepts

Collecting Information in Data Sets 2–3
Information in a Data Set 2–3
Types of Data Sets 2–4
Standard Data Set 2–5
Fixed Format 2–5
Variable Format 2–6
Compact Data Set 2–7
Direct Data Set 2–7
Random Data Set 2–8
Ordered Data Set 2–9
Unordered Data Set 2–10
Defining Data Items 2–11
Types of Data Items 2–11
Global Data Items 2–13
Data Item Options 2–14
Indexing Records of a Data Set 2–15
Keys 2–15
Sets 2–17
Subsets 2–19
Types of Sets and Subsets 2–20
Index Sequential Sets and Subsets 2–20
Index Random Sets and Subsets 2–21
Ordered List Sets and Subsets 2–22
Unordered List Sets and Subsets 2–23
Bit Vector Sets and Subsets 2–24
Connecting Data Set Records 2–25
Disadvantages of Links 2–26

Types of Links 2–26
Unprotected Link 2–27
Verified Link 2–27
Counted Link 2–28
Symbolic Link 2–29
Self-Correcting Link 2–31
Disjoint and Embedded Structures 2–33
Disjoint Structures 2–33
Embedded Structures 2–33
Restrictions on Disjoint and Embedded Structures 2–34
Physical Options 2–35
Database Parameters 2–38
For Controlling Memory Buffers 2–38
For Controlling Transactions 2–39

Section 3. Ensuring DMSII Database Security

Remaps 3–1
Logical Databases 3–3
Guard Files 3–6

Section 4. Creating a DMSII Database

Collecting and Analyzing Information 4–1
Deciding on an Implementation Approach 4–2
Using the Bottom-Up Approach 4–3
Using the Top-Down Approach 4–4
Implementing the Database for Simplicity 4–4
Using Simple Records 4–4
Using Simple Structures 4–4
Disjoint Data Sets 4–4
Foreign Key Technique 4–5
Index Sequential Sets and Automatic Subsets . . . 4–6
Turning Embedded Structures into Disjoint Structures 4–6
Choosing Relationships 4–7
One-to-One (1:1) Relationships 4–7
One-to-One (1:1) Relationships within One Data Set 4–8
One-to-One (1:1) Relationships between Two Data Sets 4–8
One-to-Many (1:M) Relationships 4–9
One-to-Many (1:M) Relationships within One Data Set 4–10
One-to-Many (1:M) Relationships between Two Data Sets 4–10
Many-to-Many (M:N) Relationships 4–11
Relationship Data Set Technique 4–12
Relationship Data Set Technique and One Data Set 4–12

Relationship Data Set Technique and Two Data Sets 4–14
Defining a Database to the System 4–15
Writing a DASDL Source File 4–16
Using ADDS Menus 4–16
Compiling Tailored Software 4–16
Initializing a Database 4–17

Section 5. Changing a DMSII Database

Updating a Database 5–1
Reorganizing a Database 5–1
Format Conversion 5–1
Garbage Collection 5–2
Modeling a Database 5–2

Section 6. Restoring a DMSII Database

Backing Up a Database 6–1
Dumping a Database 6–1
Maintaining a Dump Tape Directory 6–2
Auditing a Database 6–2
Recording a Transaction 6–3
Designating Audit Trail Attributes 6–4
Marking Quiet Points, Syncpoints, and Controlpoints . . 6–5
Quiet Points 6–5
Syncpoints 6–6
Controlpoints 6–7
Examining and Copying Audit Files 6–7
Declaring the Restart Data Set 6–7
Recovering a Database 6–8
Recovering Automatically 6–8
Abort Recovery 6–8
Halt/Load Recovery 6–8
Concurrency Control 6–9
INDEPENDENTTRANS 6–9
REAPPLYCOMPLETED 6–9
COMS Synchronized Recovery 6–10
Recovering Manually 6–11
Partial Database Recovery 6–11
Whole Database Recovery 6–12

Section 7. Accessing a DMSII Database

Extended Retrieval with Graphic Output (ERGO) 7–3
Application Program Interfaces 7–3
Language Extensions 7–3
Interpretive Interface 7–3

Section 8. Maintaining a DMSII Database

Operations Control Manager (OCM) 8–1
DBANALYZER 8–2
DBCERTIFICATION 8–2
DMMONITOR 8–3
DMUTILITY 8–3
LOADDUMP 8–3
VISIBLE DBS 8–3

Glossary 1

Bibliography 1

Index 1

Figures

1–1. DMSII Environment Functions 1–1
1–2. Interactions in Creating a Database 1–5
1–3. Interactions in Accessing a Database 1–6

2–1. Parts of a DMSII Database 2–2
2–2. Record 2–4
2–3. Standard Data Set with Fixed Format 2–5
2–4. Standard Data Set with Variable Format 2–6
2–5. Compact Data Set 2–7
2–6. Direct Data Set 2–8
2–7. Random Data Set 2–9
2–8. Ordered Data Set 2–10
2–9. Unordered Data Set 2–11
2–10. Sets 2–18
2–11. Subsets 2–19
2–12. Index Sequential Set 2–21
2–13. Index Random Set 2–22
2–14. Ordered List Set 2–23
2–15. Unordered List Sets 2–24
2–16. Bit Vector Subset 2–25
2–17. Link 2–26
2–18. Unprotected Link 2–27
2–19. Verified Link 2–28
2–20. Counted Link 2–29
2–21. Symbolic Link 2–31
2–22. Self-Correcting Link 2–32

3–1. Remap 3–2
3–2. Logical Databases 3–5
3–3. Guard File 3–7

4–1. Refining Elements and Their Relationships 4–2
4–2. Database Organization 4–3
4–3. One-to-One (1:1) Relationships 4–7
4–4. One-to-One (1:1) Relationships within One Data Set 4–8
4–5. One-to-One (1:1) Relationships between Two Data Sets 4–8
4–6. One-to-Many (1:M) Relationships 4–9
4–7. One-to-Many (1:M) Relationships within One Data Set 4–10
4–8. One-to-Many (1:M) Relationships between Two Data Sets 4–11
4–9. Many-to-Many (M:N) Relationships 4–12
4–10. Relationship Data Set and One Data Set 4–13
4–11. Relationship Data Set and Two Data Sets 4–14
4–12. Compiling Tailored Software 4–17

6–1. Audit Trail 6–4
6–2. Quiet Point 6–5

6–3. Syncpoint 6–6
6–4. Transaction Trail and Audit Trail Storage 6–10

7–1. Accessing a DMSII Database 7–2

Tables

1–1. Sources of Information on DMSII Environment Functions 1–2

2–1. Data Item Types 2–12
2–2. Physical Options 2–35

Section 1
Introducing the Data Management System II (DMSII)

DMSII is a Unisys database system that centralizes the management of information. Database structures and relationships are described only once to the system. Programs can then add, update, and retrieve data without concern for how the data is stored or organized.

Functions of the DMSII Environment

DMSII databases, software, and utilities combine to form a complete database management environment. Within this environment, you can perform many functions to ensure efficient use of your database. These functions are shown in Figure 1-1.

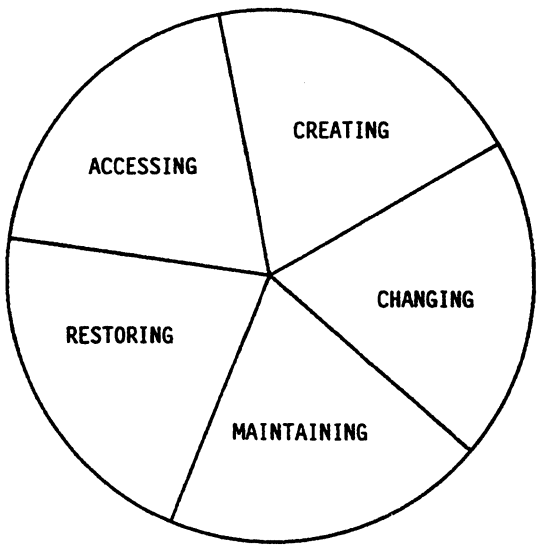

Figure 1–1. DMSII Environment Functions

Each function in Figure 1-1 is not handled by just one piece of DMSII software. Instead, different combinations of software perform each function. Table 1-1 explains the software used by each function, the section of this overview that describes the function, and the name of the manual or manuals that explain the function fully.

Table 1–1. Sources of Information on DMSII Environment Functions

Function	DMSII Software	Section	Relevant Manuals
Creating a database	DASDL and ADDS	4, "Creating a DMSII Database"	DMSII DASDL Reference Manual and InfoExec Administration Guide
Changing a database	Standard DMSII software	5, "Changing a DMSII Database"	DMSII Utilities Operations Guide
Restoring a database	Standard DMSII software and COMS	6, "Restoring a DMSII Database"	DMSII Utilities Operations Guide and COMS Operations Guide
Accessing a database	ERGO, language extensions, and interpretive interface	7, "Accessing a DMSII Database"	ERGO Operations Guide, DMSII Application Programming Guide, DMSII Interpretive Interface Reference Manual, and Volume 2 of the appropriate language manuals
Maintaining a database	OCM and DMSII utilities	8, "Maintaining a DMSII Database"	InfoExec Operations Guide and DMSII Utilities Operations Guide

DMSII Framework

To perform these functions, the DMSII environment relies on a strong framework of software and files. This framework consists of the following parts:

- Data and Structure Definition Language (DASDL)

  The language that defines DMSII databases

- Description file

  The file of tables that contain the structural characteristics of a database

- Accessroutines

  Routines that coordinate the access of a database by many users at the same time

- Control file

  The file that controls database operations

- DMSUPPORT library

  The library that contains common functions used by the Accessroutines and certain DMSII software

DASDL

DASDL is the data definition language for DMSII. You use DASDL to describe a database physically and logically. Through DASDL, you can also ensure the integrity and security of information in the database.

To create a database, you write a series of DASDL statements in a DASDL source. The DASDL compiler, called SYSTEM/DASDL, compiles this source to produce a description file.

The declarations in this overview are written in DASDL.

Description File

A description file consists of tables, which contain all the structural characteristics of a database. Each database has one description file, which is called DESCRIPTION/<database name>.

The system uses the description file as the central source of information when it compiles tailored software and user programs that access a particular database. (Tailored software is described in Section 4, "Creating a DMSII Database.") The description file contains a complete description of the database in a form readable by the compiler. It provides the compiler with the record layout of each data set.

The description file is used by the following:

- The DMALGOL compiler uses the file to produce a DMSUPPORT library tailored to the database.
- SYSTEM/DMCONTROL uses the file to produce a control file.
- SYSTEM/DBANALYZER uses the information in the descritpion file to provide an analysis of each structure.
- SYSTEM/LOADDUMP reads and interprets the description file at runtime.
- SYSTEM/PRINTAUDIT uses the description file to determine the structure number when structures are requested by name.
- SYSTEM/DBCERTIFICATION reads and interprets the description file to certify the database structures.

Accessroutines

The Accessroutines is a collection of special routines that coordinates the access of a database by many users at the same time. It manages the database, logically and physically, and is used for all basic data manipulation.

The Accessroutines, which by default is called SYSTEM/ACCESSROUTINES, is invoked by the first program to open the database. Any other programs that open the database are attached to the Accessroutines that is currently running. When the last program closes the database, the Accessroutines stops as well.

The Master Control Program (MCP) ensures that only one copy of the Accessroutines is active for each database. Using only one copy controls and synchronizes database use when several programs are accessing the database at the same time.

Control File

A control file controls database operations by doing the following:

- Controlling database interlock. This operation locks the database from changes during specific functions. Such functions need exclusive database control when they operate on the database without opening it.
- Verifying that all database data files are compatible by checking the database timestamp, version timestamp, and update level.
- Storing audit control information, dynamic database parameters, and other information.

SYSTEM/DMCONTROL both creates and maintains the control file, which is named <database name>/CONTROL.

For more information on the control file structure and use, see the DMSII Utilities Operations Guide.

DMSUPPORT Library

A DMSUPPORT library is a tailored program that contains common functions used by the Accessroutines, DMINTERPRETER, DMUTILITY, and DBCERTIFICATION. It provides entry points that allow user programs to get error codes.

For more information on tailored software, see Section 4, "Creating a DMSII Database." For more information on DMINTERPRETER, see Section 7, "Accessing a DMSII Database." For more information on DBCERTIFICATION, see Section 8, "Maintaining a DMSII Database." For more information on error codes, see the DMSII Application Programming Guide.

The DMSUPPORT library, which is called DMSUPPORT/<database name>, is generated for each database, from a source called DATABASE/DMSUPPORT.

How the DMSII Framework and Software Interact

The architecture and software of DMSII interact differently according to whether you are creating or accessing a database.

Interactions in Creating a Database

Figure 1-2 shows how the DMSII framework and software interact when you create a database.

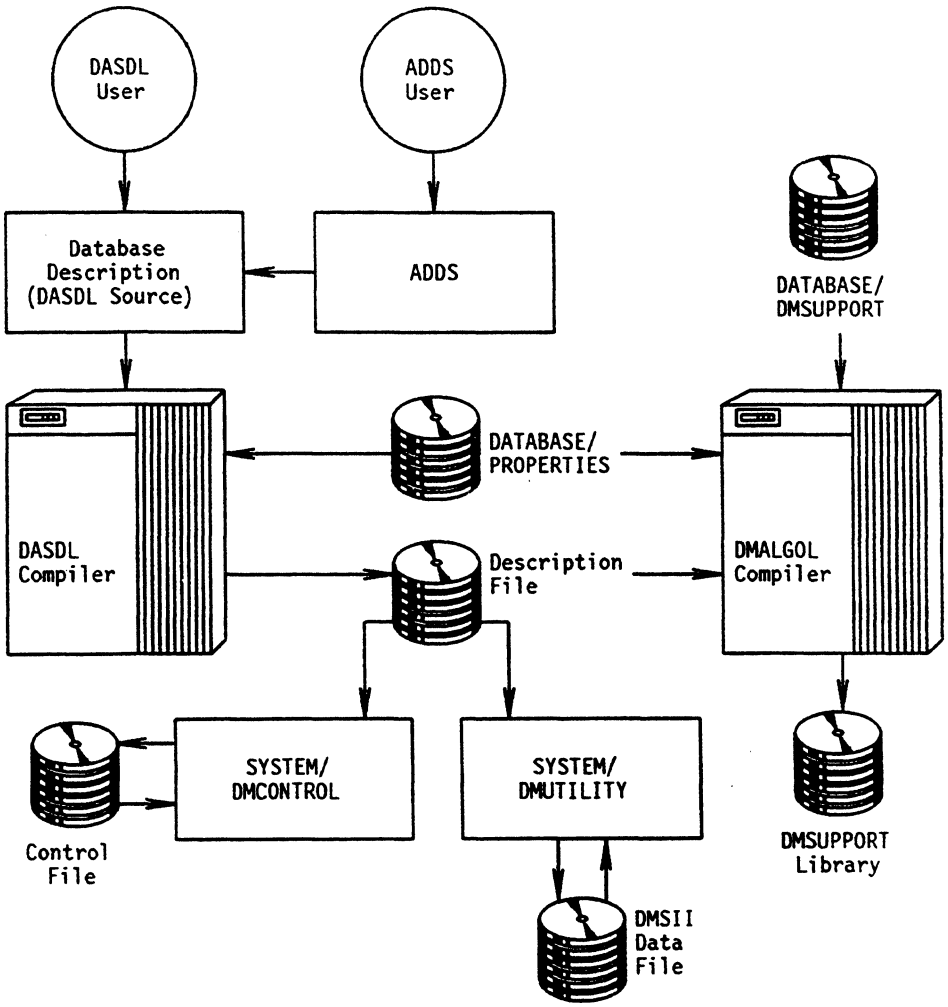

Figure 1–2. Interactions in Creating a Database

You can describe a database through DASDL or through the interactive menus of the Advanced Data Dictionary System (ADDS). In either case, the DASDL compiler takes the database definition, checks it for correctness, and produces a description file.

ADDS is described in Section 4, "Creating a DMSII Database." Partitioned databases are described in Section 2, "Understanding DMSII Database Concepts."

Interactions in Accessing a Database

Figure 1-3 shows how the DMSII framework and software interact when you access a database.

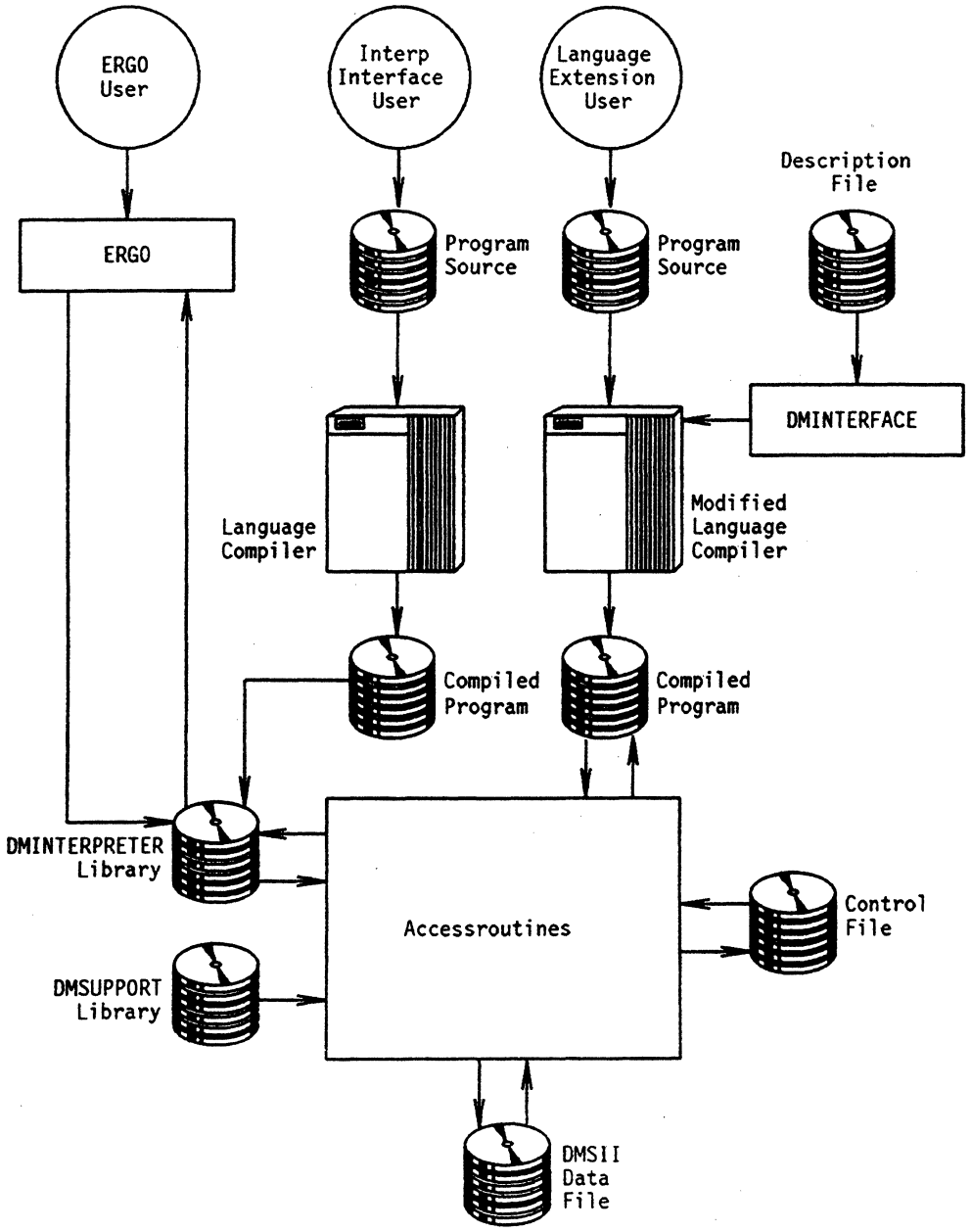

Figure 1–3. Interactions in Accessing a Database

Users can access data items interactively through ERGO or with application programs through the interpretive interface. Both methods access the database by using DMINTERPRETER library entry points.

Users can also use applications programs to access the database using language extensions. This method gets database structure information from the description file.

All access requests pass through the Accessroutines, which coordinates access of the database by using information from the DMSUPPORT library.

ERGO, the interpretive interface, and language extensions are described in Section 7, "Accessing a DMSII Database."

Section 2
Understanding DMSII Database Concepts

A DMSII database collects information into data files that can later be manipulated. The parts (or *structures*) of a DMSII database fall into the following categories:

- Data set

  A collection of data records with the same structure. Examples of data sets include EMPLOYEES, CUSTOMERS, and PROJECTS.

- Data item

  A unit of information in a database. For example, data items in the EMPLOYEES data set can include EMP-NAME, EMP-ADDRESS, and EMP-NBR.

- Global data items

  Data items that store information about the database or its structures. For example, a global data item called TOT-EMP can count the employees in the EMPLOYEES data set.

A database can index the records of a data set with the following structures:

- Set

  Retrieves all members of a data set in a designated order. For example, the EMPLOYEES data set can be ordered by name with the BY-NAME set and by employee number with the BY-EMP-NBR set.

- Subset

  Retrieves only the members of a data set that meet a designated condition. For example, the EMPLOYEES data set can include a MANAGER subset, which indexes only managers, and a CUSTOMER-REP subset, which indexes only customer representatives.

Consider a typical company. The different kinds of information that a company can manage and the way these kinds of information relate to each other correspond to the parts of a DMSII database is shown by Figure 2-1:

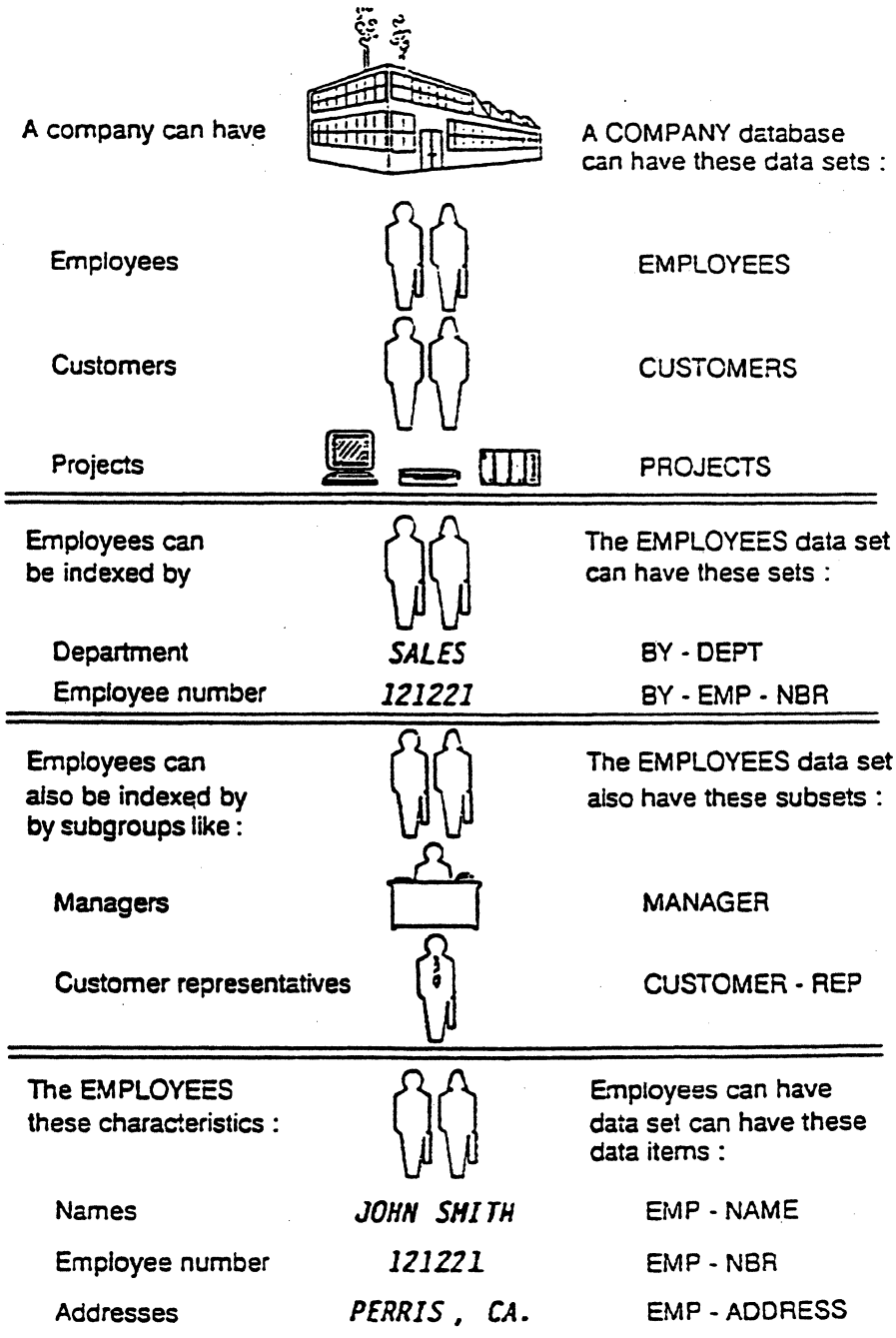

Figure 2–1. Parts of a DMSII Database

For example, the company's employees can reside in an EMPLOYEES data set. Because employees have characteristics of names and addresses, the employees data set can have data items called EMP-NAME and EMP-ADDRESS.

Employees can be indexed by department and employee number, so records in the EMPLOYEES data set can be indexed by a BY-DEPT set and BY-EMP-NBR set. Because you can classify employees into more specific subgroups, for example, managers and customer representatives, the EMPLOYEES data set can also contain the subsets MANAGER and CUSTOMER-REP.

Through a DMSII database, you can also join one data set to another, or connect members of a data set to each other through a structure called a *link*.

You can also control how database structures are physically stored on disk by using physical options and database parameters.

These structures, physical options, and parameters are further described later in this section. They are fully described in the DMSII DASDL Reference Manual.

Collecting Information in Data Sets

A data set, like a file, is a collection of data records within the same structure. However, unlike a file, a data set can collect structures and items. For example, a data set can contain other data sets, sets, and subsets, as well as data items.

The COMPANY database example can contain the following data sets:

- EMPLOYEES

  Stores information about the employees, such as names, addresses, and salaries
- CUSTOMERS

  Stores information about the customers, including company names, products ordered, and location
- PROJECTS

  Stores information about projects, such as name, cost, start date, and end date

Information in a Data Set

A typical data set, such as EMPLOYEES, can contain the following tabular information:

Name	Number	Address	Pay Type
Johnson, Bill	441234	Nyack,NY	N
Garcia, Mary	132445	Buncombe,IL	E
Chin, David	233189	Sopchoppy,FL	N
Wells, Sheila	224534	Giddings,TX	E
Endicott, Race	372451	Empire,LA	E

Name	Number	Address	Pay Type
Warren, Aldo	178556	Cocolalla,ID	N
Smith, John	121221	Perris,CA	E

Each column of the table is a *data item* and each row is a *record*. The following example represents a DASDL declaration for the previous table:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
  );
```

In this case, EMP-NAME and EMP-NBR are examples of data items. The individuals in the table, such as John Smith and his name, number, address, and pay type, are examples of records.

Each record is treated as a unit during input and output. For example, Figure 2-2 shows how information for John Smith exists in the record.

Record Address	EMP-NAME	EMP #	EMP-ADDRESS	EMP-PAY-TYPE
1	Smith, John	121221	Perris, CA	E

Figure 2–2. Record

Types of Data Sets

There are six types of data sets:

- Standard

  The default type. Use this type of data set unless you know that any of the others would improve performance over this one. For technical information on data set performance, read the DMSII DASDL Reference Manual.

- Compact

  Compacts records before storing them.

- Direct

  Stores records in a logical order defined by an unsigned numeric item.

- Random

  Stores records by using a hashing algorithm.

- Ordered

Stores records in a logical order defined by an alphanumeric item.

- Unordered

  Stores records in random order.

There is also a special kind of data set called the *restart data set*. This data set is explained under "Auditing a Database" in Section 6, "Restoring a DMSII Database."

Standard Data Set

The standard data set is the default and the one most commonly used. Standard data sets can have two formats: fixed and variable.

Fixed Format

Standard data sets with a fixed format have records of a fixed length and are the most common.

The following example declares a standard data set called EMPLOYEES. All records in this data set have an unchanging length: the length required to hold the information in EMP-NAME, EMP-NBR, EMP-ADDRESS, and EMP-PAY-TYPE.

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
  );
```

Fixed-length records are illustrated by Figure 2-3:

Record Address	EMP-NAME	EMP #	EMP-ADDRESS	EMP-PAY-TYPE
1	Smith, John	121221	Perris, CA	E
2	Warren, Aldo	178556	Cocolalla,ID	N

Figure 2–3. Standard Data Set with Fixed Format

The second record contains the same items as the first record. So, both are the same size.

Note: All types of data sets use records with fixed formats as their standard records.

Variable Format

Standard data sets with a variable format have two parts: a fixed part, which has the same length for all records, and a variable part, which changes length for different records.

The following example declares the EMPLOYEES data set to be a variable-format data set. The fixed part contains the same items across all records: EMP-NAME, EMP-NBR, EMP-ADDRESS, and EMP-PAY-TYPE. A fifth data item, EMP-JOB-TYPE, is a record-type item. Record-type items show the number of variable record types that can be declared for a variable-format data set. In this case, the EMPLOYEES data set can have two variable records: one for managers and one for general employees.

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
    EMP-JOB-TYPE          RECORD TYPE (2);  % Record-type item
  ),
   1: ( BONUS             REAL  (Ø8,Ø2);    % Record for managers
        NBR-OF-WORKERS    NUMBER(Ø3);
      ),
   2: ( PROJECT-NBR       NUMBER(Ø4);       % Record for employees
      );
```

If the EMP-JOB-TYPE item designates 1, then the record is that of a manager. The variable part of a manager record contains two items: BONUS and NBR-OF-WORKERS.

If the EMP-JOB-TYPE item designates 2, then the record is that of an employee. The variable part of an employee record contains one item: PROJECT-NBR.

The difference between the two formats is illustrated by Figure 2-4:

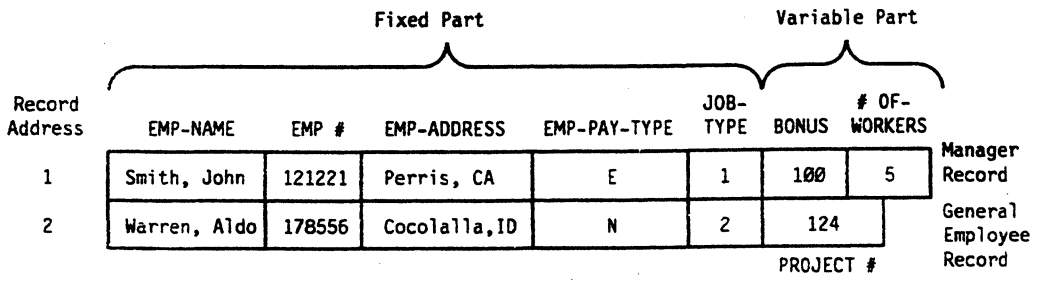

Figure 2–4. Standard Data Set with Variable Format

Because the first record is that of a manager, it contains a variable part consisting of the BONUS and NBR-OF-WORKERS items. The second record is that of a general

employee. The variable part of this record contains just one item, PROJECT-NBR, so this record is shorter than the first record.

Note: *Only standard and unordered data sets can use records with variable formats.*

Compact Data Set

Compact data sets save space by not storing trailing blanks in alphanumeric items and null items on disk. So, while records are the same size in memory, they vary in size when stored.

The following EMPLOYEES compact data set keeps track of employee dependents:

```
EMPLOYEES COMPACT DATA SET
  (
    EMP-NBR              NUMBER(Ø6) STORED OPTIONALLY;
    NBR-DEPS             NUMBER(Ø2);
    DEP-NAME             ALPHA (15) SIZE VARYING;
  );
```

This declaration is illustrated by Figure 2-5.

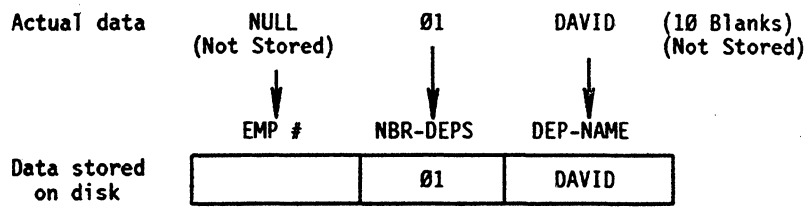

Figure 2–5. Compact Data Set

The actual data has a null EMP-NBR, a NBR-DEPS of one, and a DEP-NAME of DAVID, which consists of 10 trailing blanks.

The null item and 10 trailing blanks are eliminated before the data is stored on disk. When the record is retrieved, the null item and the trailing blanks are restored.

Direct Data Set

Direct data sets store records in the order defined by a data item that is an unsigned integer. A data item on which records are ordered and retrieved is called a *key*. (Keys are further explained under "Keys" in this section.)

The following example finds records in the DEPARTMENTS direct data set. The example declares DEPT-LIST as an Access that uses DEPT-NBR (a numeric value) as a key. An *Access* retrieves records from direct, random, and ordered data sets in the order defined by a key. Accesses do not physically exist on disk.

```
DEPARTMENTS DIRECT DATA SET
  (
    DEPT-NBR              NUMBER(04);
    DEPT-NAME             ALPHA (50);
  );

DEPT-LIST ACCESS OF DEPARTMENTS
  KEY IS DEPT-NBR;
```

This declaration is illustrated by Figure 2-6.

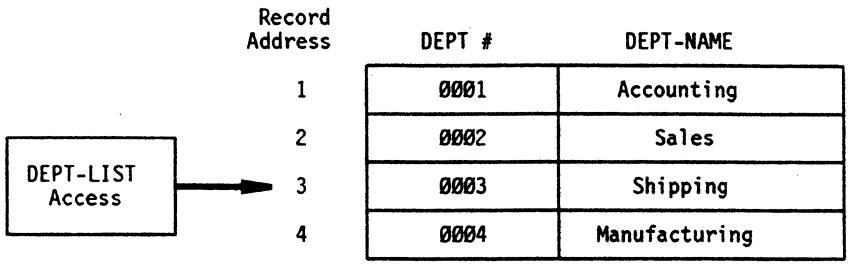

Figure 2–6. Direct Data Set

The DEPT-LIST Access uses the DEPT-NBR item as a key to directly locate a record in the DEPARTMENTS direct data set.

Random Data Set

Random data sets store records in no logical order and use an Access to retrieve records. However, the Access does not retrieve a record directly. Instead, DMSII uses a hashing algorithm to transform the Access key into a number that locates a table. The record is located in this table.

You can use the DASDL *MODULUS* option to declare the number of tables in which to store records. For information on the MODULUS option, see "Physical Options" in this section.

The following random data set declares DEPT-LIST as an Access that uses DEPT-NBR (a numeric value) as a key. The key is used in a hashing algorithm to find records in the DEPARTMENTS random data set.

```
DEPARTMENTS RANDOM DATA SET
  (
    DEPT-NBR              NUMBER(04);
    DEPT-NAME             ALPHA (50);
  );

DEPT-LIST ACCESS OF DEPARTMENTS
  KEY IS DEPT-NBR;
```

The declaration is identical to the one previously described for a direct data set. However, the hashing algorithm uses the Access to calculate the location of a record. This declaration is illustrated by Figure 2-7.

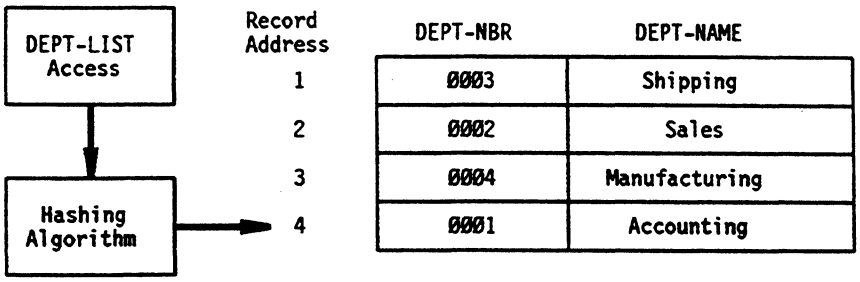

Figure 2–7. Random Data Set

The DEPT-LIST Access uses an algorithm on the DEPT-NBR key to locate a record directly in the DEPARTMENTS random data set.

Ordered Data Set

Ordered data sets maintain records in physical order based on a required key. An Access retrieves records using this key. You cannot specify sets when using ordered data sets.

The following DEPENDENTS ordered data set is contained within the EMPLOYEES data set. Data sets contained within another data set are called *embedded data sets*. Ordered data sets do not have to be embedded, but they usually are. (Embedded structures are explained near the end of this section.) Records within DEPENDENTS are ordered by the numeric DEP-AGE key in the DEP-LIST Access.

```
EMPLOYEES DATA SET
  (
    EMP-NAME                ALPHA (15);
    EMP-NBR                 NUMBER(06);
      DEPENDENTS ORDERED DATA SET
        (
          DEP-NAME          ALPHA (15);
          DEP-AGE           NUMBER(03);
        );
     DEP-LIST ACCESS TO DEPENDENTS
       KEY IS DEP-AGE
         DUPLICATES ALLOWED;
  );
```

This declaration is illustrated by Figure 2-8.

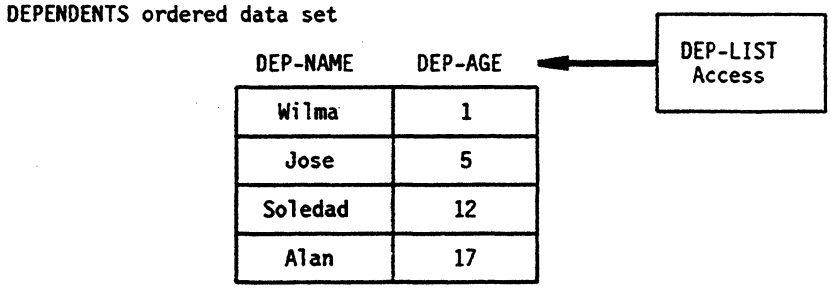

Figure 2–8. Ordered Data Set

The DEP-LIST Access uses DEP-AGE as a key to ordering one employee's records in the DEPENDENTS ordered data set.

Unordered Data Set

Unordered data sets store records in random order. Because of this, unordered data sets are seldomly used. If order is not important, you might want to use unordered data sets for efficiency purposes.

The following DEPENDENTS unordered data set is contained within the EMPLOYEES data set. Data sets contained within another data set are called *embedded data sets*. Although unordered data sets do not have to be embedded, they normally are. (Embedded structures are discussed near the end of this section.) Records within DEPENDENTS are stored in random order.

```
EMPLOYEES DATA SET
  (
    EMP-NAME                ALPHA (15);
    EMP-NBR                 NUMBER(Ø6);
      DEPENDENTS UNORDERED DATA SET
        (
          DEP-NAME          ALPHA (15);
          DEP-BIRTHDATE     NUMBER(Ø8);
        );
  );
```

Figure 2-9 shows how an unordered DEPENDENTS data set for one record might be stored.

DEPENDENTS unordered data set

DEP-NAME	DEP-BIRTHDATE
Wilma	12151987
Jose	Ø6Ø11976
Soledad	Ø4171971
Alan	Ø8281983

Figure 2–9. Unordered Data Set

Defining Data Items

A data item is a unit of information in a database record. The following example declares several data items:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
  );
```

Specifically, this EMPLOYEES data set has the following data items:

Data Item Name	Meaning	Data Example
EMP-NAME	Employee name	John Smith
EMP-NBR	Employee number	121221
EMP-ADDRESS	Employee address	Perris, CA
EMP-PAY-TYPE	Employee type of pay	E for exempt

Types of Data Items

Almost all information can be described with the eight types of data items: alphanumeric, numeric, real, Boolean, field, group, count, and filler. These types are detailed in Table 2-1.

Table 2–1. Data Item Types

Data Item	Explanation	Example

Alpha item	Represents words and characters such as names, addresses and titles. Data is stored as EBCDIC characters, or as Kanji characters if USAGE=KANJI is specified.	EMP-NAME ALPHA (15);
		EMP-NAME can have the following values:
		`Smith, John`
		`Garcia, Mary`
		`Johnson, Bill`

Numeric item	Represents integers and fractions, with or without signs. A numeric item declaration can have precision (number of places to the right of the decimal point), a scale factor (number of digits in an item), and a sign.	EMP-NBR NUMBER(06); EMP-NBR can have the following values: 121221 132445 441234

Real item	Contains single-precision floating-point numbers and occupies one word.	EMP-SALARY REAL (08,02);
		EMP-SALARY can have the following values:
		`324.45`
		`129.25`
		`8ØØ.99`

Boolean item	Represents TRUE and FALSE values and takes up one 4-bit digit of storage. Only the rightmost bit contains information.	RETIRED BOOLEAN; RETIRED can have a value of 0 (FALSE) or 1 (TRUE).
Field item	Stores Boolean values more efficiently. Normally, a Boolean value takes up 4 bits of space. In a field item, Boolean values take up 1 bit of space. So, a field item can store up to 48 Boolean values. Field items can also store an unsigned, nonnegative integer that is up to 48 bits long.	WORKS-ON FIELD (DISK-PROJECT; PRINTER-PROJECT; TAPE-PROJECT; MEMORY-PROJECT;);
Group item	Consists of other alphabetic or numeric items.	EMP-NAME GROUP (FIRST-NAME ALPHA (15); MIDDLE-NAME ALPHA (01); LAST-NAME ALPHA (20););

continued

Table 2–1. Data Item Types (cont.)

Data Item	Explanation	Example
Count item	Used in conjunction with counted lines. Contains a binary integer value which indicates the number of counted links which refer to the record. The system automatically adjusts this value.	WORKERS COUNT(03);
Filler item	Reserves space in a record for future information. Filler items enable you to add new data items without reorganizing the database.	FILLER SIZE 12;

Global Data Items

Global data items store information about the entire database or any of its structures. Such data items, however, exist in a single global record independent of all other data base structures.

The following are global data items:

- Population

  Counts the maximum number of records in a data set. For example, the following declares TOT-EMP as a counter for the EMPLOYEES data set. TOT-EMP designates 10,000 as the maximum number of records.

  ```
  TOT-EMP POPULATION (10000) OF EMPLOYEES;
  ```

- Aggregate item

  Counts or sums data items in a data set. For example, the following declares TOT-SALARY as the sum of all salaries in the EMPLOYEES data set. The final total can have 12 digits, with two to the right of the decimal point.

  ```
  TOT-SALARY AGGREGATE (12,02) SUM (SALARY) OF EMPLOYEES;
  ```

  The following example uses TOT-EXEMPT to count exempt employees in the EMPLOYEES data set. An employee is exempt if the value of EMP-PAY-TYPE is equal to *E*. The final count can have five digits.

  ```
  TOT-EXEMPT (05) COUNT (EMP-PAY-TYPE = "E" ) OF EMPLOYEES;
  ```

  Aggregate items are not limited to global data items. They can be used with other data items as explained under "Embedded and Disjoint Structures" in this section.

- Data item

Can be used as a global item. For example, you can declare a numeric item called HIGHEST as a global data item. Later, you can use it to store something like the highest salary of an employee. HIGHEST can contain a number with eight digits, with two to the right of the decimal point.

```
HIGHEST                    REAL  (Ø8,Ø2);
```

Data Item Options

You can initialize or restrict the information stored in data items, including global data items, with the following options:

- NULL

  Assigns a null value to an item – that is, identifies the item as containing no valid information. The following example puts a null value of a blank space into the alphanumeric item COLLEGE-DEGREE:

```
COLLEGE-DEGREE             ALPHA (15)
  NULL " ";
```

- INITIALVALUE

  Assigns a value to a newly created item. If the item is numeric or real, the value can be a number, an integer, LOW-VALUE, or HIGH-VALUE. LOW-VALUE represents 0 for both real and numeric items. For real items, HIGH-VALUE is the largest integer that can be represented. For numeric items, HIGH-VALUE turns on all bits.

  The following example assigns a value of 0 (FALSE) to the Boolean item RETIRED:

```
RETIRED                    BOOLEAN
  INITIALVALUE Ø;
```

- REQUIRED

  Ensures that an item is assigned a value. If an item to be stored has a null value, the record with that item is not stored and you receive an error message. The following example ensures that the EMP-NAME data item is required:

```
EMP-NAME                   ALPHA(15)
  REQUIRED;
```

- OCCURS

  Shows that the item is a collection of items, each having identical attributes. Using this option on a data item is similar to using a subscripted variable. The following example declares DEP-NAME to be an alphanumeric item that occurs 10 times:

```
DEP-NAME                   ALPHA (15)
  OCCURS 1Ø TIMES;
```

- SIZE

Allows for the storage of variable length alphanumeric items. This option conserves disk space. The following example discards trailing blanks from EMP-NAME values:

```
EMP-NAME                    ALPHA(15)
  SIZE VARYING;
```

- STORED

  Compacts records stored on disk by eliminating items with a null value or items that you designate should not be stored. This option conserves disk space. The following example stores addresses only if they are not null:

```
EMP-ADDRESS                 ALPHA(20)
  STORED OPTIONALLY;
```

- VERIFY

  Stores a record into a data set only if its data item values meet designated conditions. Although this option is used with data items, its results apply to the entire data set. The following example stores a record into the EMPLOYEES data set only if the EMP-SALARY of that record is greater than 15000:

```
EMPLOYEES DATA SET
  (
    EMP-NAME                ALPHA (15);
    EMP-NBR                 NUMBER(06);
    EMP-ADDRESS             ALPHA (20);
    EMP-SALARY              REAL  (08,02);
  )
VERIFY (EMP-SALARY > 15000);
```

  Attempting to store an EMP-SALARY value less than or equal to 15000 gives you an error.

Indexing Records of a Data Set

Sets and subsets allow for the access of the records of a data set in key order. This arrangement makes record access more efficient.

Through DMSII sets or subsets, you can order data set records in many different ways based on one or more keys. For example, different sets can order records of the same employee data set using employee names, salaries, or addresses as keys.

The sets, subsets, and keys you use dictate the most efficient retrievals of data in a data set. To design the most efficient retrievals, you must know how information is retrieved. Because each set increases the expense of updating a database, a database should not contain all the sets and keys that represent all the ways records can be ordered. Only you can decide if the added set is worth the update and storage expense.

Keys

A key is the data item on which records of a set, subset, or Access are retrieved and ordered. The following example uses keys:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
  );

BY-NUMBER SET OF EMPLOYEES
  KEY IS EMP-NBR;

BY-ADDRESS SET OF EMPLOYEES
  KEY IS EMP-ADDRESS;
```

The two keys used to retrieve and order records in the EMPLOYEES data set are:

- The EMP-NBR data item is a key in the BY-NUMBER set.
- The EMP-ADDRESS data item is a key in the BY-ADDRESS set.

To ensure the integrity of entries into sets, subsets, or Accesses, you can define keys with the following options to restrict record membership for index random, index sequential, and ordered list sets and subsets. (These types of sets and subsets are explained later in this section.)

- NO DUPLICATES

  Used by default if you omit NO DUPLICATES or DUPLICATES (except for automatic subsets, for which the default is DUPLICATES). Only entries with unique values are stored. An entry with a key value that already exists is not stored.

  Key values in an existing record are changed by deleting and recreating the record, or by using the KEYCHANGEOK option of NO DUPLICATES. You can use KEYCHANGEOK only if the INDEPENDENTTRANS option was set for the database. For more information on the INDEPENDENTTRANS option, see Section 6, "Restoring a DMSII Database."

- DUPLICATES

  Allows entries to be duplicated. By default, duplicate keys are not maintained in any order. With DUPLICATES FIRST, duplicate keys are stored so that the most recently added duplicate is the first to be returned during access. With DUPLICATES LAST, duplicate keys are stored so the most recently added duplicate is the last to be returned during access.

  When using this option, you need to process all the duplicates (either through a program or interactively) to find the desired record.

Keys arrange entries in ascending or descending order by using the following options for indexed sequential and ordered list sets and subsets. (These types of sets and subsets are explained later in this section.)

- ASCENDING

  By default, puts records with low key values before records with higher key values – for example, 12345 is put before 33333.

- DESCENDING

  Puts records with high key values before records with low key values – for example, 33333 is put before 12345.

The following example ensures that the records of the BY-NUMBER set contain no duplicates with keys that can be changed and are arranged in descending order:

```
BY-NUMBER SET OF EMPLOYEES
  KEY IS EMP-NBR,
    NO DUPLICATES KEYCHANGEOK,
      DESCENDING;
```

Sets

A set contains one entry for each record in the data set. In general, this entry contains the address of the record and the data item used as a key. Sets can be declared for all types of data sets except for ordered data sets.

Sets enable you to index the same data set in several different ways. The following example shows how the EMPLOYEES data set can be indexed by two different sets:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
  );

BY-NUMBER SET OF EMPLOYEES
  KEY IS EMP-NBR;

BY-ADDRESS SET OF EMPLOYEES
  KEY IS EMP-ADDRESS;
```

This declaration is illustrated by Figure 2-10.

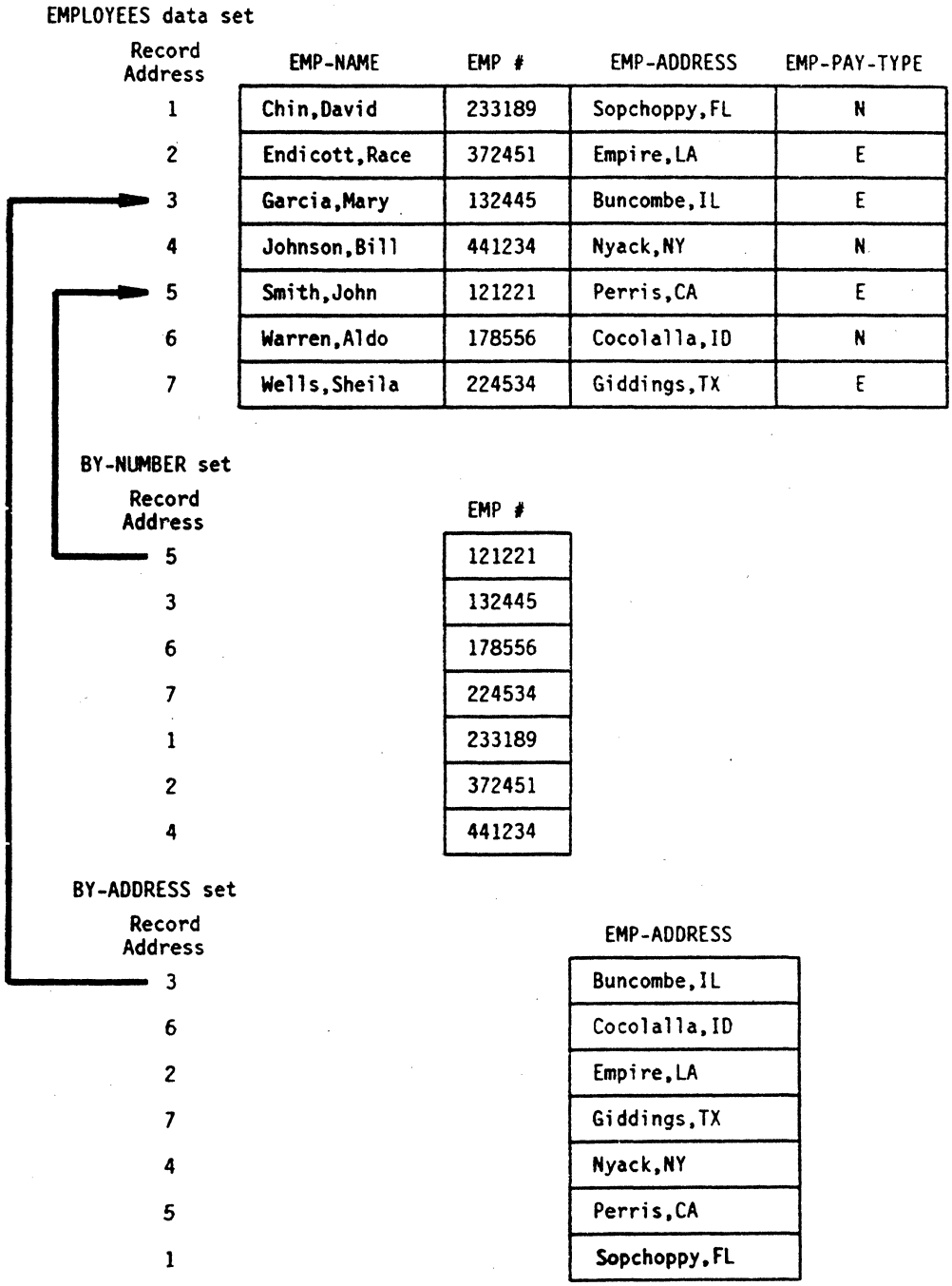

Figure 2-10. Sets

The BY-NUMBER set arranges records using the employee number as a key. The BY-ADDRESS set arranges records using the address as a key.

Subsets

A subset is like a set but it indexes only the members of the data set that meet certain conditions. In the following example, the EXEMPT subset indexes only those records with a pay type of E (exempt):

```
EXEMPT SUBSET OF EMPLOYEES WHERE EMP-PAY-TYPE = "E"
  KEY IS EMP-NAME
    ASCENDING;
```

This declaration is illustrated by Figure 2-11.

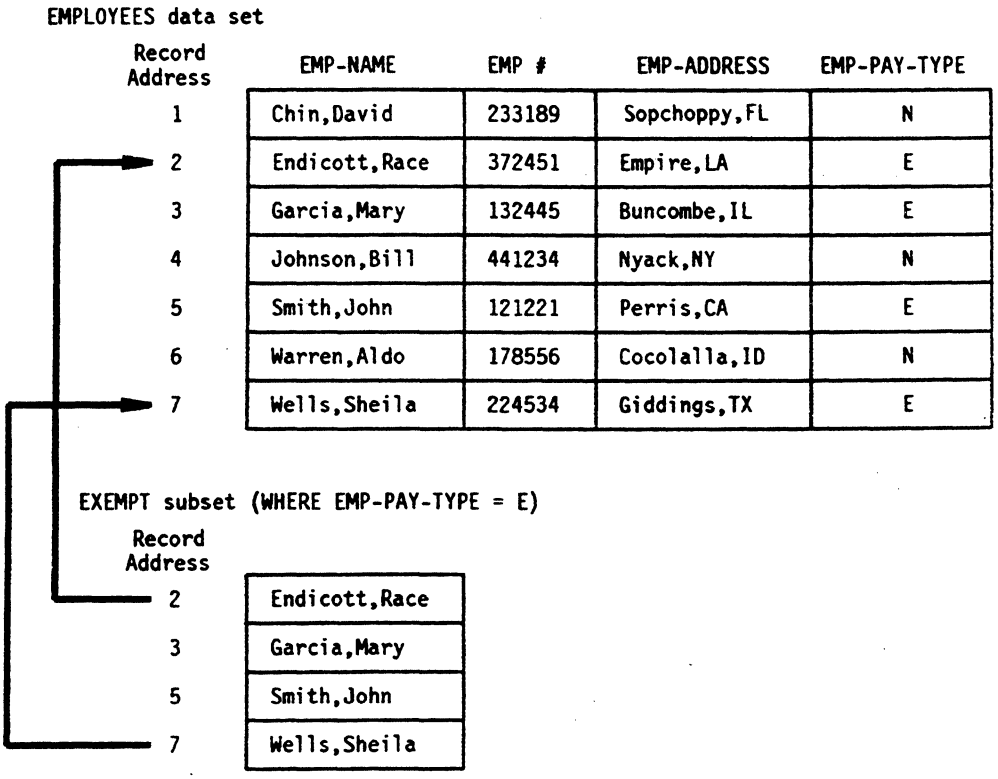

Figure 2-11. Subsets

Only four members of the EMPLOYEES data set have a pay type of E (exempt). As a result, the subset has only four records compared to the seven records of the data set. A subset normally has fewer records than the data set it indexes.

Records in a subset can be maintained automatically or manually as follows:

- Automatic subset

  DMSII automatically maintains automatic subsets by adding or deleting entries based on conditions you declare with the subset. (Such conditions are designated with a WHERE clause.) The previously described EXEMPT subset is an example of an automatic subset.

- Manual subset

  This subset does not automatically maintain its entries because it does not designate any conditions. (It does not have a WHERE clause.) Application programs need to maintain entries in manual subsets; if not, data integrity problems can result.

  For example, assume that every month certain employees receive a bonus and other perks. However, from month to month those employees change.

  You can declare a manual subset called BONUS:

```
BONUS SUBSET OF EMPLOYEES
  KEY IS MONTH,EMP-NAME
    ASCENDING;
```

  Then, you can use an application program to insert and delete records from the subset based on conditions you designate in the program.

Types of Sets and Subsets

There are five types of sets and subsets:

- Index sequential

  The default type for disjoint sets and subsets with a key. Retrieves records by a key.

  Use this type unless you know that any of the others would improve performance over this one. That knowledge requires an understanding of the technical details described in the DMSII DASDL Reference Manual.

- Index random

  Retrieves records by using a hashing algorithm on a key.

- Ordered list

  The default type for embedded sets and subsets with a key. Retrieves records in the order designated by a key.

- Unordered list

  The default type for disjoint or embedded sets and subsets without a key. Retrieves records by their addresses and not by a key.

- Bit vector

  Stores one bit for each data record.

Index Sequential Sets and Subsets

Index sequential sets and subsets consist of tables. Entries in each table can point to other tables or can contain the key values that point directly to a corresponding record in

a data set. Entries in these tables are arranged in key order to make a binary search on the key efficient.

The following example declares the BY-NUMBER set to be index sequential:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-SALARY            REAL  (Ø8,Ø2);
  );

BY-NUMBER SET OF EMPLOYEES
  KEY IS EMP-NBR,
    INDEX SEQUENTIAL;
```

Figure 2-12 shows how the initial entries of BY-NUMBER point to tables containing record addresses. These tables, in turn, point to the appropriate records in the EMPLOYEES data set.

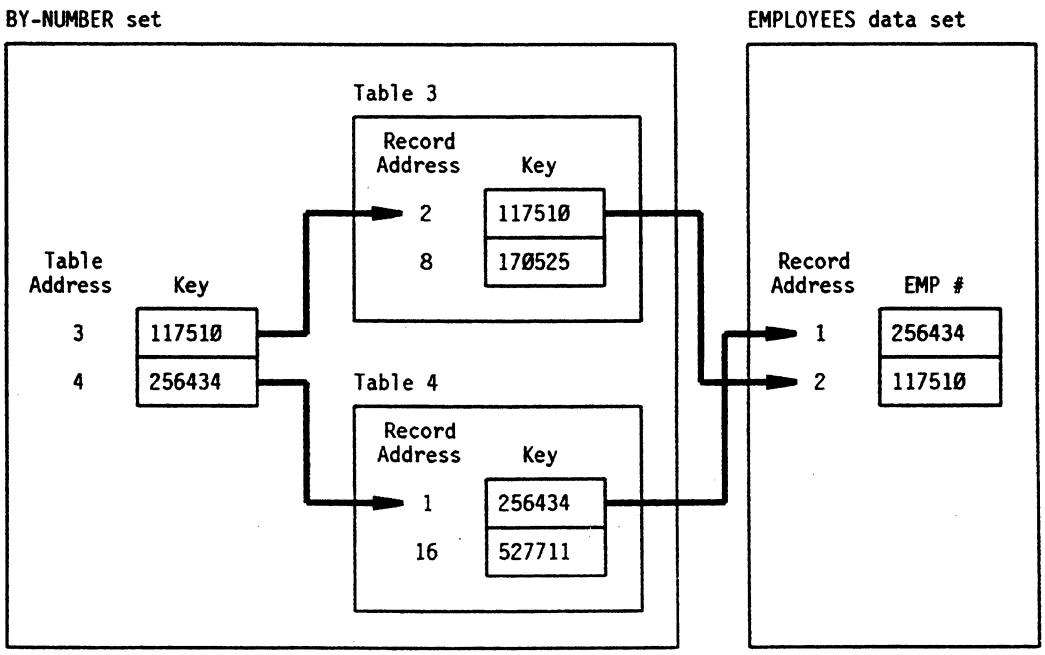

Figure 2–12. Index Sequential Set

Index Random Sets and Subsets

Like index sequential sets and subsets, entries in an index random set or subset also exist in a table. However, these tables do not maintain their entries in key order. Instead, the set or subset uses a hashing algorithm on the initial entry to compute the address of the table that contains the key and record address,

The following example declares the BY-NUMBER set to be index random:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-SALARY            REAL  (Ø8,Ø2);
  );

BY-NUMBER SET OF EMPLOYEES
  KEY IS EMP-NBR
    ASCENDING,
      INDEX RANDOM;
```

Figure 2-13 shows how the initial entries of the BY-NUMBER set are transformed by a hashing algorithm to point to tables containing record addresses. These tables, in turn, point to the appropriate records in the EMPLOYEES data set.

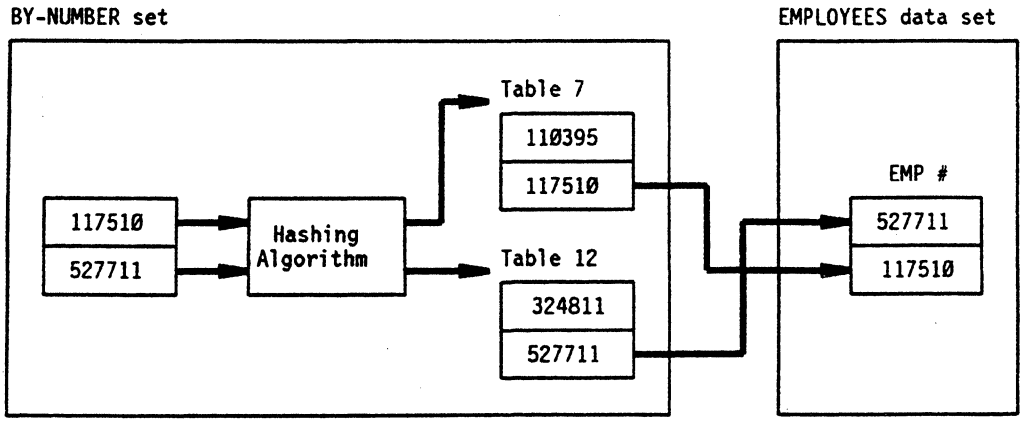

Figure 2–13. Index Random Set

Ordered List Sets and Subsets

Ordered list sets and subsets sort records by key.

The following example declares BY-NUMBER to be an ordered list set.

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-SALARY            REAL  (Ø8,Ø2);
  );

BY-NUMBER SET OF EMPLOYEES
  KEY IS EMP-NBR
    ASCENDING,
      ORDERED LIST;
```

Figure 2-14 shows how records in the BY-NUMBER set are ordered by EMP-NBR. Each record is an index into the corresponding record of the EMPLOYEES data set.

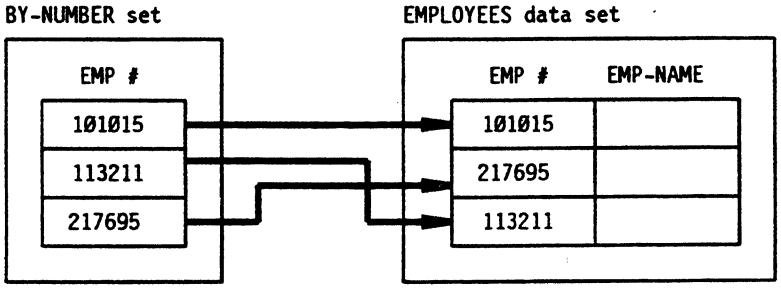

Figure 2–14. Ordered List Set

Unordered List Sets and Subsets

Unordered list sets and subsets are similar to ordered list sets and subsets. However, this type of set or subset stores the record addresses, not the key values. Records are arranged in the order in which they exist in the data set.

The following example declares DEP-LIST to be an unordered list set on the DEPENDENTS data set, which is contained within the EMPLOYEES data set. (Data sets contained within another data set are called embedded data sets. Embedded structures are explained near the end of this section.)

```
EMPLOYEES DATA SET
  (
     EMP-NBR      NUMBER(Ø6);
       DEPENDENTS DATA SET
         (
           DEP-NAME     ALPHA (15);
           DEP-AGE      NUMBER(Ø3);
         )
       DEP-LIST SET OF DEPENDENTS,
         UNORDERED LIST;
  );
```

Figure 2-15 shows how the record addresses are stored in the DEP-LIST set. The addresses locate records in the DEPENDENTS data set.

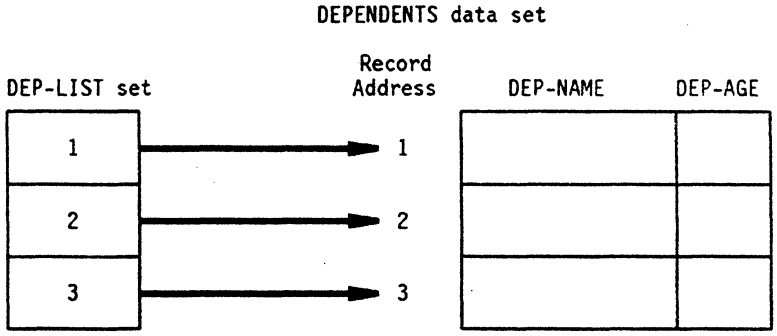

Figure 2–15. Unordered List Sets

Bit Vector Sets and Subsets

The bit vector type can be used only with standard, fixed-format data sets. It is a collection of bits where one bit represents a data set record.

Bit vector sets represent ON/OFF (TRUE/FALSE) conditions. A bit is ON (TRUE) if a record belongs in the set or subset. It is OFF (FALSE) if the record does not belong or does not exist.

The following example stores an ON in the LOW-SAL-EMPS subset for every employee whose salary is under 10000, and an OFF for employees whose salaries are greater than or equal to 10000:

```
EMPLOYEES DATA SET
  (
     EMP-NBR              NUMBER(Ø6);
     EMP-NAME             ALPHA (15);
     EMP-SALARY           NUMBER(Ø8,Ø2);
  );

LOW-SAL-EMPS SUBSET OF EMPLOYEES
  WHERE EMP-SALARY LESS THAN 1ØØØØ
    BIT VECTOR;
```

Figure 2-16 shows how each bit in the LOW-SAL-EMPS subset is turned off if the corresponding record in the EMPLOYEES data set has an EMP-SALARY greater than 10000. Conversely, the bit is turned on if the record has an EMP-SALARY less than 10000.

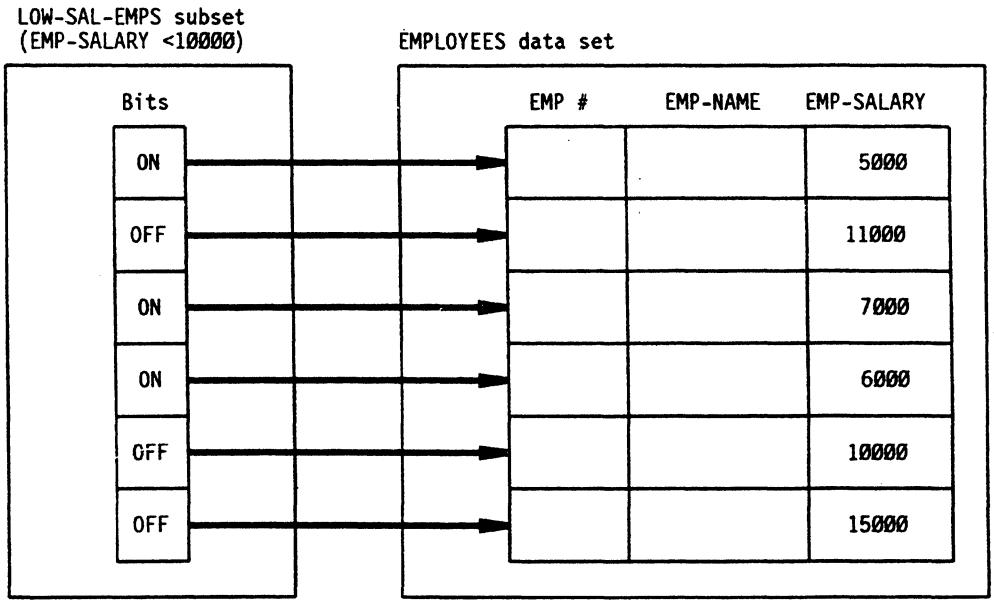

Figure 2–16. Bit Vector Subset

Connecting Data Set Records

Links connect records in one data set to records in another or the same data set.

In the following example, the EMP-TO-PROJ link joins the EMPLOYEES data set to the PROJECTS data set:

```
EMPLOYEES DATA SET
  (
    EMP-NBR                 NUMBER(06);
    EMP-NAME                ALPHA (15);
    EMP-TO-PROJ IS IN PROJECTS;
  );

PROJECTS DATA SET
  (
    PROJ-NBR                NUMBER(05);
    PROJ-NAME               ALPHA (20);
  );
```

Figure 2-17 shows how each EMP-TO-PROJ link is part of a record in the EMPLOYEES data set. Each link connects a record in the EMPLOYEES data set to a record in the PROJECTS data set.

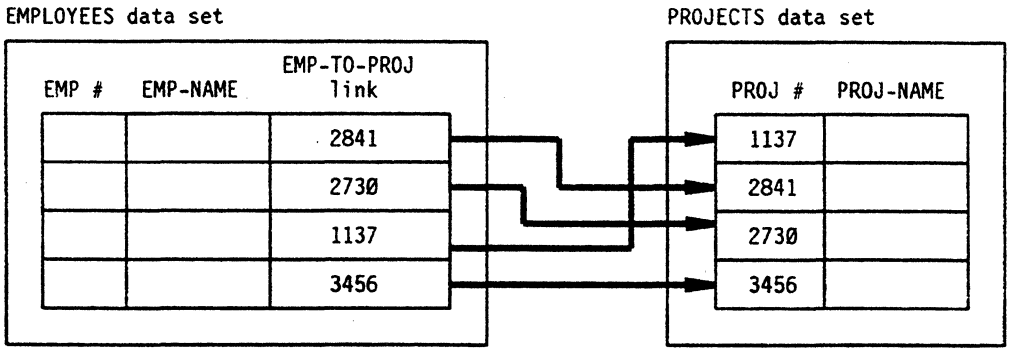

Figure 2–17. Link

Disadvantages of Links

Links can easily join unrelated records. However, they can also complicate the database as follows:

- Links must be maintained by an application program. Other DMSII structures, such as automatic subsets, can do what links do but are maintained by the system.
- If you delete a record pointed to by several links, you might forget to remove all the links pointing to that record. As a result, the links would point to nothing.
- Links are one-way pointers to a record. So, while you can find the record a link is pointing to, you cannot easily find what record is pointing to the linked record.

Alternatives to using links are discussed under "Using Simple Structures" in Section 4, "Creating a DMSII Database."

Types of Links

There are five types of links:

- Unprotected

  Stores only the address of the linked record.
- Verified

  Stores both the address of the linked record and the value of a data item in that record.
- Counted

  Counts the number of links pointing to the record.
- Symbolic

  Stores only the key value of the linked record.
- Self-correcting

Stores both the address and the key value of the linked record.

Unprotected Link

This is the simplest and the most unsafe type of link. It contains only the address of the linked record.

Your application program is responsible for ensuring that the correct record is retrieved. DMSII does not ensure that the link is pointing to the correct record if that record is moved or deleted.

The following example declares an unprotected link in the EMPLOYEES data set that points to a record in the PROJECTS data set:

```
EMPLOYEES DATA SET
  (
    EMP-NAME                ALPHA (15);
    EMP-NBR                 NUMBER(Ø6);
    EMP-TO-PROJ IS IN PROJECTS WITH NO PROTECTION;
  );

PROJECTS DATA SET
  (
    PROJ-NBR                NUMBER(Ø5);
    PROJ-NAME               ALPHA (2Ø);
  );
```

Figure 2-18 shows how the EMP-TO-PROJ link of records in the EMPLOYEES data set points to the addresses of records in the PROJECTS data set.

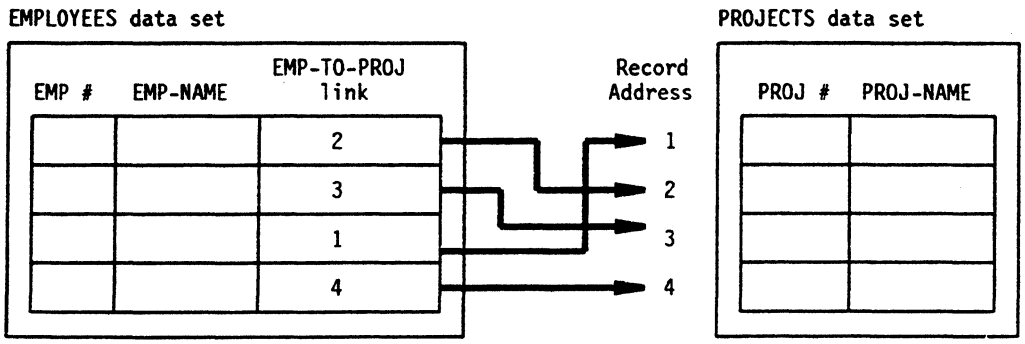

Figure 2–18. Unprotected Link

Verified Link

DMSII can ensure that a link retrieves the correct record through a verified link. This type of link stores both the address of the linked record and the value of a data item in that record. The data item is designated after the words *VERIFY ON*.

When retrieving the record, DMSII compares the value of the item stored with the link to the value of the item in the record located using the address. If they are not equal, DMSII returns an error.

The following example verifies the EMP-TO-PROJ link using the PROJ-NBR data item of the PROJECTS data set:

```
EMPLOYEES DATA SET
  (
    EMP-NBR                NUMBER(Ø6);
    EMP-NAME               ALPHA (15);
    EMP-TO-PROJ IS IN PROJECTS
      VERIFY ON PROJ-NBR;
  );

PROJECTS DATA SET
  (
    PROJ-NBR               NUMBER(Ø5);
    PROJ-NAME              ALPHA (2Ø);
  );
```

Figure 2-19 shows how records in the EMPLOYEES data set link to records in the PROJECTS data set through the EMP-TO-PROJ verified link. The link contains the record address and the value of PROJ-NBR in the linked record. For example, when the record at record address 2 is retrieved, the value of the item (4435) is compared to the value stored with the link (4435). In this case, they are equal and no error is returned.

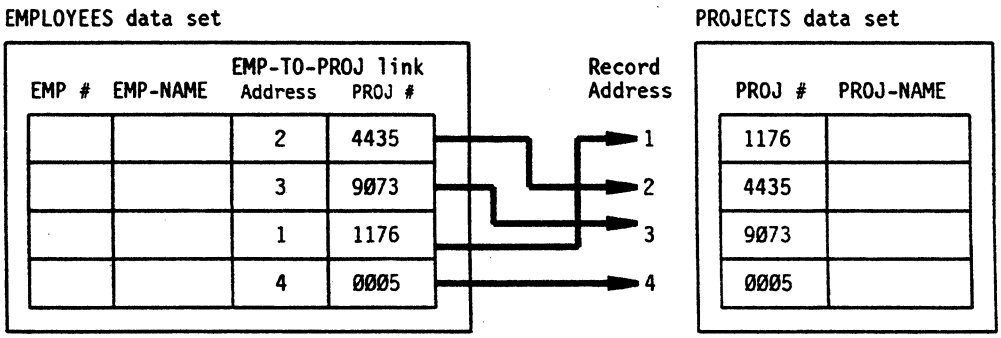

Figure 2–19. Verified Link

Counted Link

A counted link uses a record item to count the number of links that point to the record and to designate a maximum count. If the count is greater than 0, the record cannot be deleted.

In the following example, the EMP-TO-PROJ link points to the PROJECTS data set, with the number of links counted by WORKERS. If the value in WORKERS is greater

than 0, a project cannot be deleted. WORKERS can have a maximum of three records pointing to it at the same time.

```
EMPLOYEES DATA SET
  (
    EMP-NBR                NUMBER(Ø6);
    EMP-NAME               ALPHA (15);
    EMP-TO-PROJ IS IN PROJECTS
      COUNTED;
  );

PROJECTS DATA SET
  (
    PROJ-NBR               NUMBER(Ø5);
    PROJ-NAME              ALPHA (2Ø);
    WORKERS                COUNT (Ø3);
  );
```

Figure 2-20 shows that three records from the EMPLOYEES data set are linked to record 2 on the PROJECTS data set through the EMP-TO-PROJ link. The WORKERS item in the PROJECTS data set counts three links.

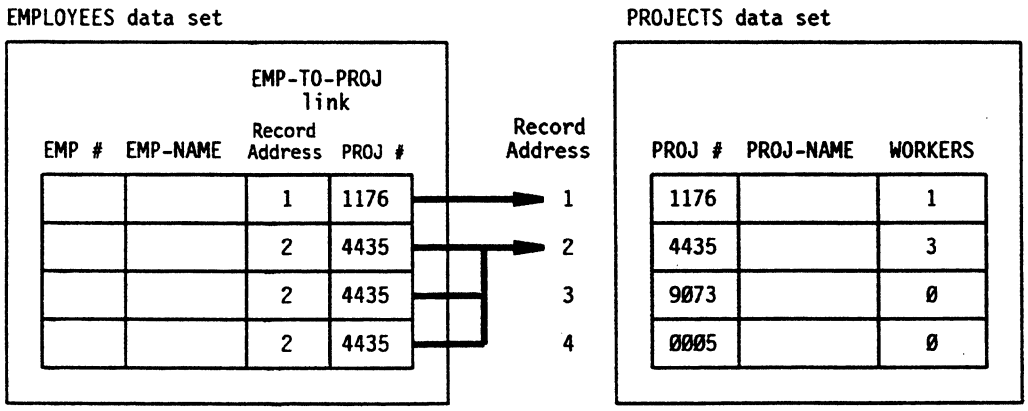

Figure 2–20. Counted Link

Symbolic Link

A symbolic link stores the key value of the linked record. When the link retrieves the record, it uses this key value to search the set.

In the following example, the OWNPROJECTS symbolic link uses the BY-NAME set to search the PROJECTS data set:

```
EMPLOYEES DATA SET
  (
    EMP-NBR                 NUMBER(Ø6);
    EMP-NAME                ALPHA (15);
    OWNPROJECTS IS KEY OF BY-NAME;
  );

PROJECTS DATA SET
  (
    PROJ-NBR                NUMBER(Ø5);
    PROJ-NAME               ALPHA (2Ø);
  );

BY-NAME SET OF PROJECTS
  KEY IS PROJ-NAME
    DESCENDING,
      NO DUPLICATES,
        ORDERED LIST;
```

Figure 2-21 shows how the OWNPROJECTS link first searches the BY-NAME set for the key PRINTER-PROJECT, and then uses that key to find the correct record in PROJECTS data set.

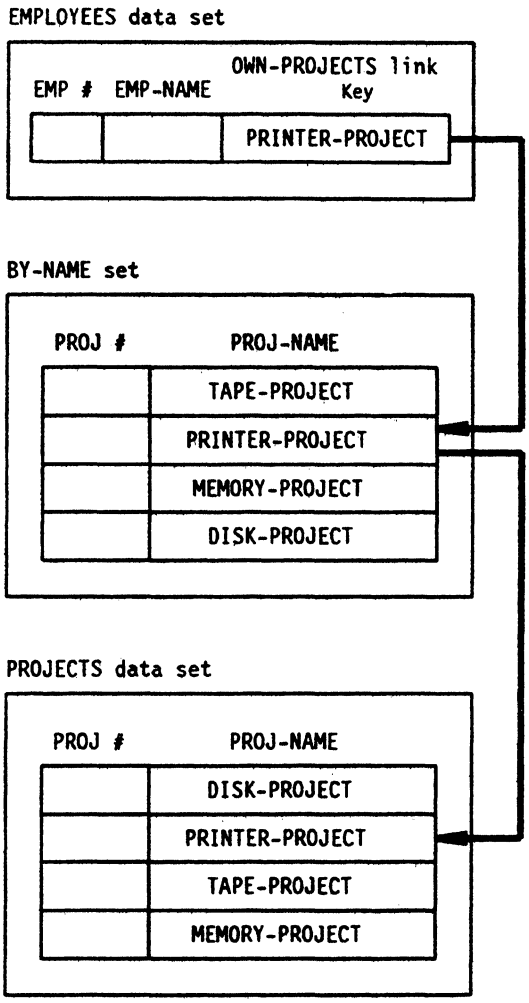

Figure 2–21. Symbolic Link

Self-Correcting Link

Like an unprotected link, a self-correcting link stores the address of the linked record. However, like a symbolic link, the self-correcting link also stores the key value of the linked record.

When retrieving a record, the self-correcting link gets the record at the stored address. If the value of that record does not match the value stored by the link, the link searches for the correct record using the key. When the correct key is found, the correct address is placed in the link item.

In the following example, the OWNPROJECTS self-correcting link uses the BY-NAME set to search the PROJECTS data set:

```
EMPLOYEES DATA SET
  (
    EMP-NBR                NUMBER(Ø6);
    EMP-NAME               ALPHA (Ø5);
    OWNPROJECTS IS IN BY-NAME;
  );

PROJECTS DATA SET
  (
    PROJ-NBR               NUMBER(Ø5);
    PROJ-NAME              ALPHA (2Ø);
  );

BY-PROJ SET OF PROJECTS
  KEY IS PROJ-NAME
    DESCENDING,
      NO DUPLICATES,
        ORDERED LIST;
```

Figure 2-22 shows how a record in the EMPLOYEES data set links to a record in the PROJECTS data set through the OWNPROJECTS link. The OWNPROJECTS link points directly to the correct record using its stored record address. If the record has moved from that address, the link searches the BY-NAME set to find the correct key for the record.

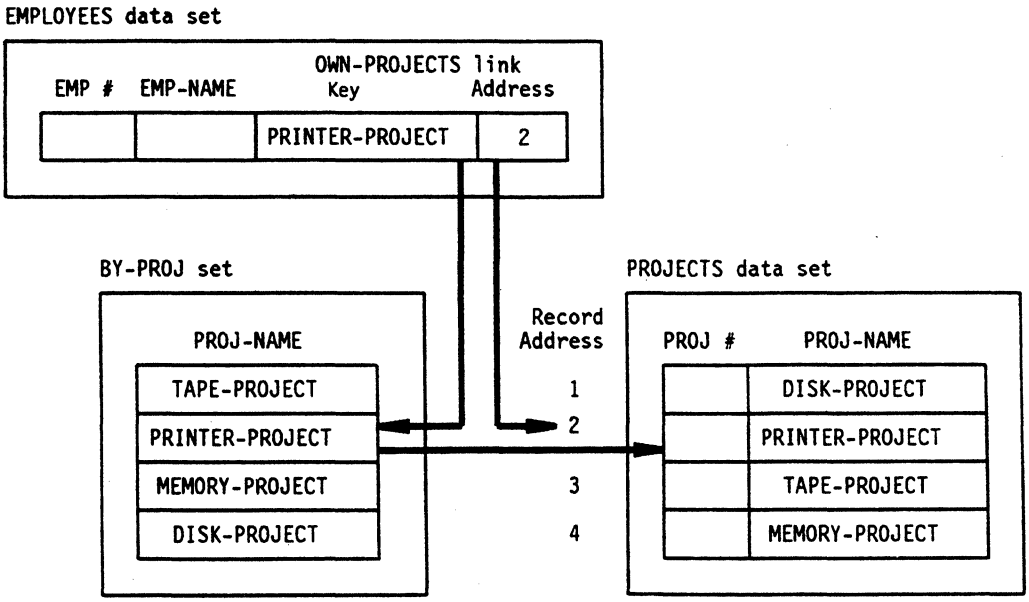

Figure 2–22. Self-Correcting Link

Disjoint and Embedded Structures

Data sets, sets, and subsets can exist in two states: *disjoint* or *embedded*. These two kinds of structures are described in the following text.

For further information on using these structures, see "Using Simple Structures" in Section 4, "Creating a DMSII Database."

Disjoint Structures

Disjoint structures stand on their own and are not contained within other structures. You use disjoint structures to show independent information. For example, the following EMPLOYEES and CUSTOMERS data sets are disjoint:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
  );

CUSTOMERS DATA SET
  (
    CUST-NAME             ALPHA (15);
    CUST-NBR              NUMBER(Ø5);
    CUST-ADDRESS          ALPHA (2Ø);
  );
```

The EMPLOYEES and CUSTOMERS data sets do not depend on any other structures in the database.

Use disjoint structures whenever possible. You can easily represent them using tables and you can maintain information on only one data set at a time. For example, to change a customer name, you can go through the CUSTOMERS data set without disturbing the EMPLOYEES data set.

Embedded Structures

Embedded structures are contained within a data set and depend on it. You use embedded structures to show subordinate relationships. For example, the following DEPENDENTS data set is embedded in the EMPLOYEES data set:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
      DEPENDENTS ORDERED DATA SET
        (
          DEP-NAME        ALPHA (15);
          DEP-AGE         NUMBER(Ø3);
        );
  );
```

The DEPENDENTS data set cannot stand on its own. Its information depends on information in the EMPLOYEES data set. In this case the EMPLOYEES data set is the *master data set* because the DEPENDENTS data set is subordinate to it.

Embedded structures can complicate the data base. Changing information in one embedded structure involves going through at least one other master data set. For example, to change information in the DEPENDENTS data set, you must go through the EMPLOYEES data set first.

Restrictions on Disjoint and Embedded Structures

All the previously described data sets, sets, and subsets can be disjoint or embedded with the following restrictions:

- Ordered data sets are almost always embedded but can be disjoint.
- Direct data sets can only be disjoint.
- Index random sets and subsets can only be disjoint.
- Bit vector sets can only be disjoint.
- If a master record contains a key, all embedded records must be empty before you can delete the master record.
- Ordered list is the default type for embedded sets and subsets with a key.
- Index sequential is the default type for disjoint data sets.
- An aggregate item can be used with embedded data sets. In such a case, the item is declared in the master structure to count or sum items in embedded data sets.

The following example declares an aggregate item called TOTDEP that counts the number of dependents in the DEPENDENT data set, which is embedded in the EMPLOYEES data set. The number 2 after the word AGGREGATE shows that this item can store up to a 2-digit item.

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    TOTDEP AGGREGATE (2) COUNT (DEP-NAME) OF DEPENDENTS;
      DEPENDENTS DATA SET
        (
          DEP-NAME        ALPHA (15);
          DEP-AGE         NUMBER(Ø3);
        )
  );
```

Physical Options

Data sets, sets, subsets, data items, and links are structures that define how a database is organized logically.

These logical structures correspond to records that are physically stored on disk. You can define how to store these records using physical options. Physical options are designated in the declarations for databases, data sets, sets, and subsets.

The physical options in Table 2-2 are explained further in the DMSII DASDL Reference Manual. This table names the option, lists the structures for which the option can apply, and describes the option.

Table 2–2. Physical Options

Option	Structure	Description
ADDRESSCHECK	Special	Adds a word to all data blocks of every database file. This word is used to ensure that DMSII is reading the correct block. This special option does not apply to specific structures, but to the entire database. This option uses the following format: `OPTIONS(ADDRESSCHECK);`
ALLOCATE	Data set	Controls how available space is allocated for new records in unordered data sets.

continued

Table 2–2. Physical Options (cont.)

Option	Structure	Description
AREAS, AREASIZE	Data set, set, and subset	Defines how disk files are divided. AREAS defines the maximum number of areas in which a file is divided. AREASIZE defines the size of an area. If these options are not designated, DMSII can calculate the values of these options by using a designated POPULATION option.
AVERAGE RECORD SIZE	Data set	Defines the average size of records in compact data sets.
BLOCKSIZE	Data set	Controls the size of the block, in which records in a data set, set, or subset are normally grouped.
BUFFERS	Global default, data set	Controls the number of buffers the system can allocate for a structure and the additional buffers the system can allocate for each program that invokes a structure.
CHECKSUM	Global default, data set	Detects I/O errors. When the system stores a block, it stores a checksum value. When the system reads the block, it recomputes the checksum value and compares it to the stored value. If both values do not match, the block has an error.
DIGITCHECK	Data set	Ensures that all required numbers of a data set contain decimal digits.
DISK/PACK	Global default, data set	Controls the location of database files.
LOADFACTOR	Set and subset	For index random, index sequential, and ordered list sets and subsets only. Controls the number of entries moved when tables are split.
LOCK TO MODIFY DETAILS	Global default, data set	Ensures that user programs lock the master record before adding, deleting, or changing records belonging to the master. This option guarantees that two programs do not change records of the same master at the same time.
MEMORY RESIDENT	Data set, set, and subset	Keeps buffers in memory rather than on disk to save processing time.

continued

Table 2–2. Physical Options (cont.)

Option	Structure	Description
MODULUS	Set, subset, Access	For index random sets and subsets only and the only physical option that applies to Accesses. Controls the number of tables allocated when the set, subset, or Access is initialized.
PARTITIONS, OPEN PARTITIONS	Data set	PARTITIONS designates the partitioned data set, and OPEN PARTITIONS designates how many different partitions of a structure can be used at the same time. Both options apply to partitioned structures only. Normally, all records of a data set reside in a single file. However, data sets that would require very large files are instead partitioned among several files. This partitioned database reduces the amount of disk or pack space required if only a small part of the database is used at any one time. Note that you cannot add a partitioned structure to an existing database that is not partitioned.
POPULATION	Data set	Defines the expected size of a data set. For disjoint data sets, POPULATION shows the maximum number of records that a data set can contain. For embedded data sets, POPULATION shows the average number of records per master.
PRIME	Set, subset	Designates a set or subset as the primary index to a compact, standard, or unordered data set and that the set or subset is necessary for reorganization. This attribute is used by the reorganization process only.
REBLOCK, REBLOCKFACTOR	Global default, data set	Optimizes both random and serial access to a structure. REBLOCK controls whether the Accessroutines uses reblocking. The REBLOCKFACTOR is a number defining the degree of REBLOCK.
SECURITYGUARD	Global default, data set	Defines the name and location of the guard file. Guard files are explained in Section 3, "Ensuring Database Security."
SUBBLOCKSIZE	Data set	Divides blocks within embedded ordered data sets into two or more subblocks.
TABLESIZE	Data set, set, and subset	Controls the size of the tables into which set and subset entries are grouped.

Global defaults designate default values for the entire database or for structures in the database such as data sets, sets, and subsets. These values hold true until you override them. Default values designated for structures override values designated by global defaults.

Examples of Physical Options in Declarations

- The following example declares global defaults for the options CHECKSUM, PACK, and LOCK TO MODIFY DETAILS:

```
DEFAULTS  ( CHECKSUM = TRUE,
            PACK = MYPACK,
            LOCK TO MODIFY DETAILS = TRUE
          );
```

- The following example defines default values for the options POPULATION, DIGITCHECK, and MEMORY RESIDENT for the data set:

```
DEFAULTS ( DATA SET (
                      POPULATION = 3500,
                      DIGITCHECK = TRUE,
                      MEMORY RESIDENT = ALL
                    )
         );
```

- The following example defines the options MEMORY RESIDENT, AREAS, and BLOCKSIZE for the BY-NUMBER set:

```
BY-NUMBER SET OF EMPLOYEES
KEY IS EMP-NBR,
   MEMORY RESIDENT = ALL,
   AREAS = 100,
   BLOCKSIZE = 100;
```

Database Parameters

You use database parameters to designate the values used for memory buffers and certain database operations.

For Controlling Memory Buffers

Database parameters control how database information is stored in memory buffers before being written to disk. These parameters include the following:

- ALLOWEDCORE

  Designates the maximum amount of memory that DMSII can use for database buffers

- OVERLAYGOAL

  Controls the rate at which buffers are written to disk. The default is 5 percent of the value of ALLOWEDCORE per minute

Other parameters of this kind include SHAREDALLOWEDCORE, LOCALALLOWEDCORE, and RESIDENT LIMIT, which are explained in the DMSII Utilities Operations Guide.

The following example declares an ALLOWEDCORE value of 100 words and an OVERLAYGOAL value of 5 percent of the ALLOWEDCORE value.

```
PARAMETERS
 (
  ALLOWEDCORE = 1ØØ,
  OVERLAYGOAL = 5
 );
```

For Controlling Transactions

The database SYNCPOINT and CONTROLPOINT parameters control how often certain sequences of database operations called *transactions* happen. Transactions are explained under "Auditing a Database" in Section 6, "Restoring a DMSII Database."

Section 3
Ensuring DMSII Database Security

The total database that you store on a medium such as a disk is called the *physical database*. Programs and users do not normally access the physical database for two reasons:

- A physical database does not provide enough security for its records. For example, within the COMPANY database an employee who can access the PROJECTS data set, for which he or she has authority, can also access the payroll records in the EMPLOYEES data set, for which he or she has no authority.
- Data in physical databases is not independent of the programs that use the data. When the record formats in a physical database are changed, any programs using those data sets must be recompiled.

To avoid these problems, DMSII provides a limited view of a database and its records through the following features:

- Remap

  A logical data set that redefines the physical data set
- Logical database

  A logical collection of structures that redefines and provides a view of the physical database

These logical views of the database are the ones that programs and users normally access. Through logical structures, users and programs access only those parts of the database that they are authorized to see without affecting any other part of the physical database.

Logical structures also provide *data independence*. If the logical structures remain the same, you can change the record formats of the physical database without having to recompile the programs accessing that database.

DMSII provides additional security through the use of guard files, which restrict logical or physical database access to certain users or programs.

These security features are explained in the following pages.

Remaps

A remap is a different view of a physical data set. Remaps can change the order and names of data set items and add new verification rules.

The following example shows how the PROJECTS data set is remapped to a CURRENT-PROJECTS data set:

```
PROJECTS DATA SET
  (
    PROJ-NAME       ALPHA (20);
    PROJ-NBR        NUMBER(04);
    START-DATE      ALPHA (08);
    END-DATE        NUMBER(08);
  );

CURRENT-PROJECTS SUBSET OF PROJECTS
  WHERE END-DATE > 19890101;

CURRENT-PROJECTS REMAPS PROJECTS READONLY ALL, REQUIRED ALL;
  (
    CURRENT-NAME  = PROJ-NAME;
    PROJ-NBR;
    START-DATE;
    END-DATE;
  )
  SELECT END-DATE > 19890101;
```

Figure 3-1 illustrates this declaration.

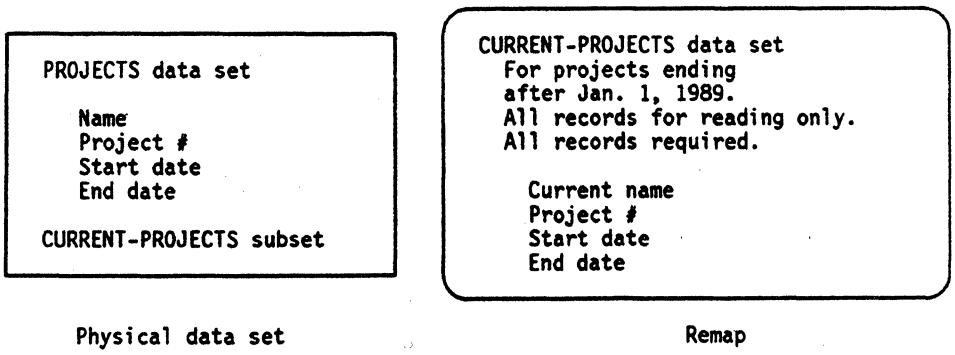

Figure 3–1. Remap

The following differences exist between the remapped data set CURRENT-PROJECTS and the physical data set PROJECTS:

- The CURRENT-PROJECTS data set includes current projects only – that is projects that meet the SELECT condition: END-DATE > 19890101. So, the remapped data set returns fewer records than the physical data set.

  Remaps can also use the VERIFY option to designate a condition for verification. The VERIFY option is described under "Data Item Options" in Section 2, "Understanding DMSII Database Concepts."

- The remapped data set designates the following options:

  – READONLY ALL

    Ensures that all items from that data set can only be read and not changed

  – REQUIRED ALL

    Ensures that all items have a value that is not null

  Remaps can also designate the following options for individual data items: INITIALVALUE, OCCURS, and REQUIRED. These options are described under "Data Item Options" in Section 2, "Understanding DMSII Database Concepts." An additional option is HIDDEN, which makes an item inaccessible and eliminates it from the user work area.

- The PROJ-NAME data item was renamed as CURRENT-NAME.
- The CURRENT-PROJECTS subset was eliminated from the CURRENT-PROJECTS data set.

Data items from both data sets are otherwise identical.

Logical Databases

Just as a remap is a different view of a physical data set, a logical database is a different view of a physical database. This view restricts some users from accessing certain database structures and controls how those structures can be used.

For example, the COMPANY database can have a logical database for the sales department and a logical database for general use, as in this declaration:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-SALARY            REAL  (Ø8,Ø2);
  );

BY-NAME SET OF EMPLOYEES
  KEY IS EMP-NAME;

MANAGER SUBSET OF EMPLOYEES;

GENERAL-EMPLOYEES REMAPS EMPLOYEES
  (
    EMP-NAME;
```

```
    EMP-NBR;
    EMP-ADDRESS;
    EMP-SALARY              HIDDEN;
  );

PROJECTS DATA SET
  (
    PROJ-NAME               ALPHA (20);
    PROJ-NBR                NUMBER(04);
    START-DATE              NUMBER(08);
    END-DATE                NUMBER(08);
  );

CURRENT-PROJECTS SUBSET OF PROJECTS
  WHERE END-DATE > 19890101;

%=============================================================================%
%                   Remaps to PROJECT Data Set                                %
%=============================================================================%

CURRENT-PROJECTS REMAPS PROJECTS
  READONLY ALL,
    REQUIRED ALL;
  (
   CURRENT-NAME  = PROJ-NAME;
    PROJ-NBR;
    START-DATE;
    END-DATE;
  ) SELECT END-DATE > 19890101;

FINISHED-PROJECTS REMAPS PROJECTS;
  (
    FINISHED-NAME  = PROJ-NAME;
    PROJ-NBR;
    COMPLETION-DATE = END-DATE;
  ) SELECT COMPLETION-DATE < 19890101;

%========================== end remaps ================================%

CUSTOMERS DATA SET
  (
    CUST-NAME               ALPHA (15);
    CUST-NBR                NUMBER(05);
    CUST-ADDRESS            ALPHA (20);
  );

%=============================================================================%
%          Logical databases for general employees                            %
%           and the sales department                                          %
%=============================================================================%

GENERAL DATABASE  (
```

```
                    GENERAL-EMPLOYEES (SETS BY-NAME, MANAGER),
                    CURRENT-PROJECTS
                  );

SALES DATABASE    (
                    CUSTOMERS,
                    FINISHED-PROJECTS
                  );
```

Figure 3-2 illustrates this declaration:

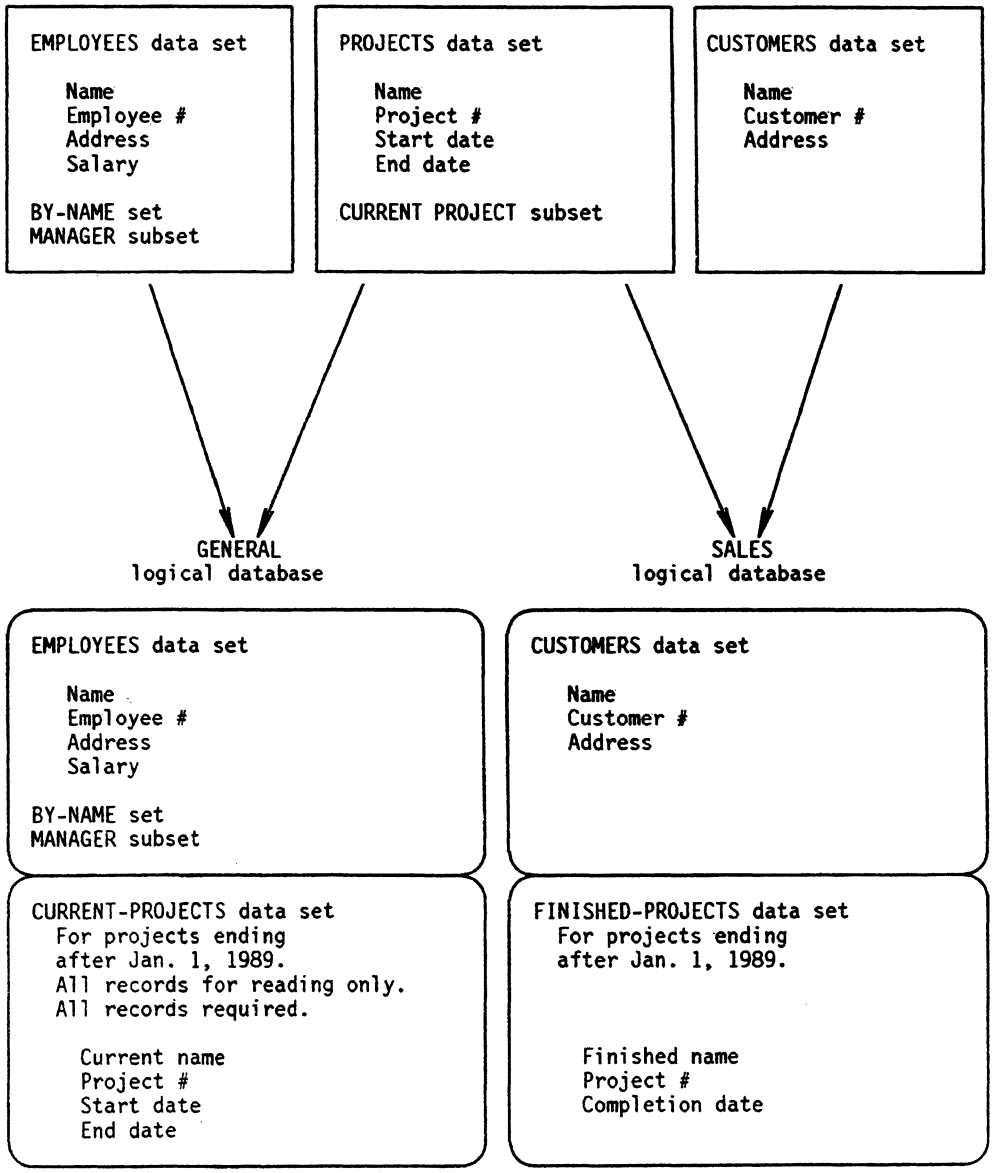

Figure 3–2. Logical Databases

In this example, two groups of users can use the physical COMPANY database through two logical databases:

- All employees can access only the EMPLOYEES data set and PROJECTS data set, which was remapped to CURRENT-PROJECTS. They can only read and not change records from these data sets.
- The sales department can access only the CUSTOMERS data set and the PROJECTS data set, which was remapped to FINISHED-PROJECTS.

Programs and users can use only the structures contained in the logical database that they open. For example, someone from the sales department cannot access records from the EMPLOYEES data set of the full COMPANY data base.

Guard Files

To further restrict access to either physical or logical databases, the GUARDFILE utility can designate the name of a guard file. This file designates the authorized users of a database or database files. The guard file can restrict access by usercode, program name, or a combination of the two.

Guard files can also restrict access to all DMSII language extension statements using different combinations of usercodes and programs. For example, you can allow the PAYROLL usercode to retrieve and update records in the EMPLOYEES data set while allowing the PERSONNEL usercode to retrieve records only.

Figure 3-3 illustrates a typical guard file.

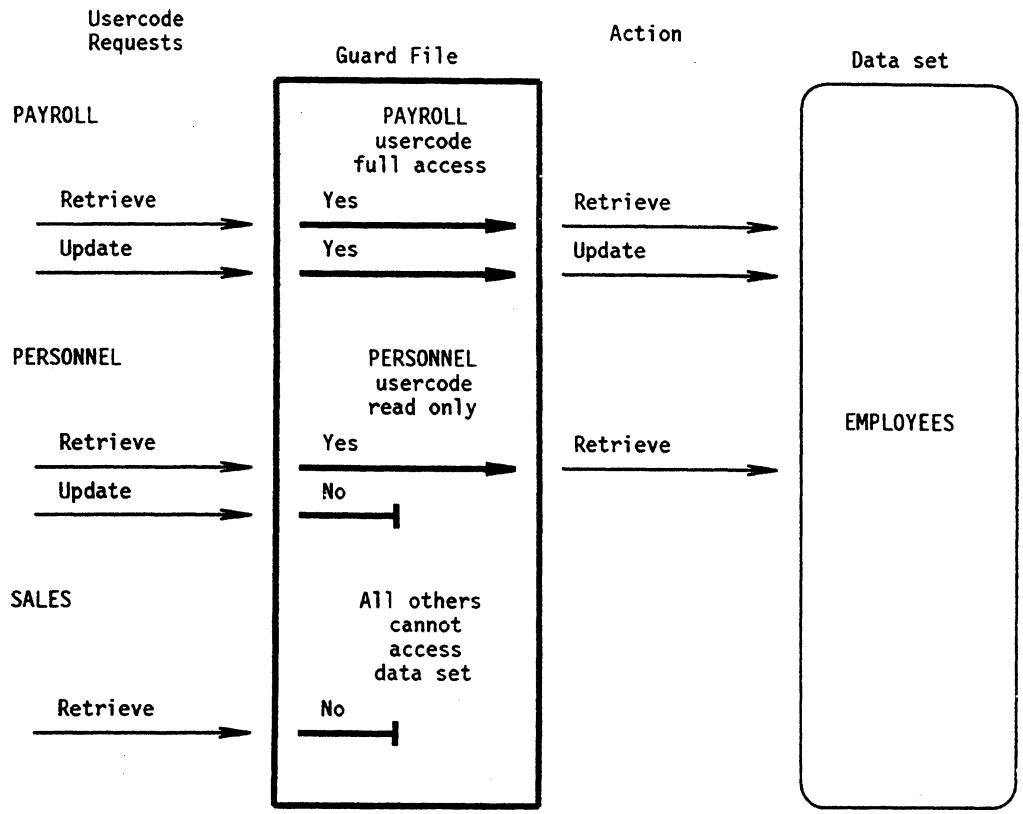

Figure 3–3. Guard File

In this case, the guard file allows full access to the EMPLOYEE data set by the PAYROLL usercode, retrieval access by the PERSONNEL usercode, and no access by any other usercode. So, the guard file allows all requests by PAYROLL and retrieval requests by PERSONNEL to go through. It denies all other requests.

When declaring a database, you can designate the guard file name in two ways:

- GUARDFILE

  Designates the name and location of a guard file that uses the Accessroutines to control access to database files.

- SECURITYGUARD

  Designates the name and location of a guard file that uses programs other than the Accessroutines to control access to database files

For more information on guard files, see the Security Features Guide. For more information on the GUARDFILE and SECURITYGUARD options, see the DMSII Utilities Operations Guide.

Section 4
Creating a DMSII Database

Creating a DMSII database involves the following steps:

1. Collecting and analyzing information
2. Deciding on an approach
3. Implementing the database for simplicity
4. Choosing relationships
5. Mapping information into a database
6. Defining the database to the system
7. Compiling tailored software
8. Initializing the database

These steps are described in this section.

Collecting and Analyzing Information

By organizing your data on paper, you can often anticipate and avoid design problems before they reach the system. Problems on paper are easy to correct. Problems in the system may not be.

Here are five guidelines for collecting and analyzing information:

1. Define and analyze actions.

   Study your documents, your employees' tasks, and anything else that you use in your current processing system. You can then determine which actions are fundamental to your business.

   For example, some actions in a typical company can include developing projects, promoting employees, and selling to customers.

2. Build scenarios.

   A scenario is a sequence of events that can identify all the elements and relationships in a particular action. An element is anything that can be stored in the database; for example, employee, department, and product are elements.

   For example, the action *developing projects* can identify elements such as the projects, the employees, and the manager. Developing this action into a scenario produces even more elements:

   - A manager of a specific department creates projects to fulfill department goals.
   - The manager allocates materials, time, and projects to teams of employees.
   - The employees develop their projects.

3. Break down scenarios.

   Separate a scenario into its parts and list those parts as they appear. To dissect a scenario, look for elements that are linked by verbs. The verbs generally show a relationship.

   For example, the clause *The manager allocates projects to teams of employees* can have three parts: manager, employees and projects. The phrase *allocates to* defines the relationship between the manager, the projects, and the employees.

4. Classify elements and identify relationships.

   By grouping similar items, you find the relationships between individual elements.

   The scenario on developing projects can identify two groups of people – employees and managers – and a group of projects. It also identifies some relationships: employees work under a manager, a manager controls several projects, and several employees can work on a project.

5. Refine the elements and their relationships.

   You can sometimes combine groups of elements or split a group into other groups. Combining or splitting groups changes the relationships between individual elements.

   For example, because managers are also employees, the managers group can be combined with the employees group. Within the employees group, managers are identified by the word *manager*. Relationships now exist between the manager elements and the employee elements in the employees group. Figure 4-1 illustrates these relationships.

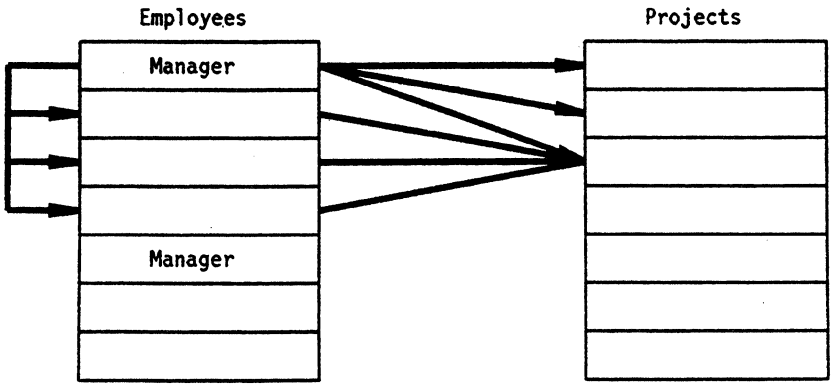

Figure 4–1. Refining Elements and Their Relationships

Deciding on an Implementation Approach

After collecting and analyzing information, you can decide how best to approach the job of structuring that information within a DMSII database.

DMSII databases organize data sets, data items, and records as shown in Figure 4-2.

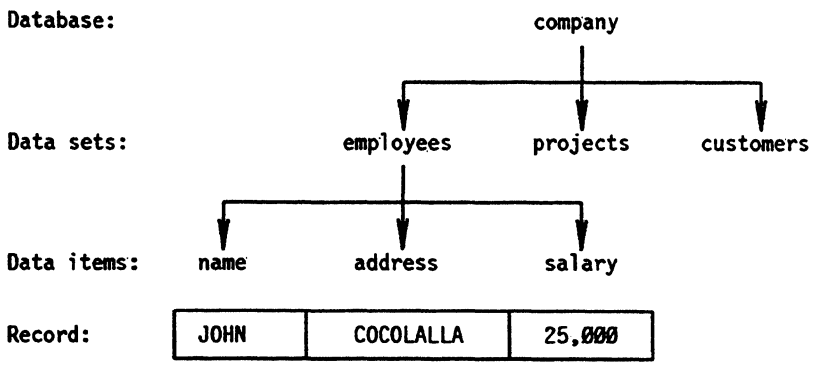

Figure 4–2. Database Organization

A database is the highest level of organization in DMSII. It contains data sets, which in turn contain data items. The data items describe the lowest level of organization: records.

You can create this arrangement in two ways: the bottom-up approach or the top-down approach. These approaches are explained in the following text.

Using the Bottom-Up Approach

The bottom-up approach describes the database from the lowest level, the record, up to the highest level, the database.

For example, you begin with each individual element of information—such as an employee named John that corresponds to a record. John has characteristics—a name, an address, and a salary—that correspond to the data items in the record. John is a member of a group of individuals called employees, which corresponds to a data set. Finally, groups of elements such as employees, projects, and customers can be organized into one database.

This approach works well for simple situations. However, defining a database from the bottom up fails when the individual has many characteristics. When you begin to collect information, you might not know all the characteristics of each individual element. If you define the higher levels of the database only when you have completely defined the individual elements, you might take a long time to define your database.

The approach also fails if many relationships exist between characteristics and individuals. For example, assume that a project has several employees but each employee can only have one project. Designing from the bottom up encourages you to define employees separately from projects.

Separating the definitions might inadvertently cause you to define both employees and projects as having several relationships with each other. As a result, although each project is correctly defined as having several employees, each employee is incorrectly defined as having several projects.

Using the Top-Down Approach

The top-down approach avoids the problems of designing from the bottom up. The top-down approach describes the database from the highest level, the database, to the lowest, the record.

For example, you begin with the largest organization, the company, which corresponds to a database. A company has several groups, including employees, projects, and customers, that correspond to data sets. Group members, such as employees, have characteristics of names, addresses, and salaries that correspond to data items. Finally, an individual member such as John corresponds to a record.

The top-down approach is generally a better approach because you can completely define the higher levels of the database, such as groups of individuals. You can then clearly see the relationships between the groups.

You can then define individuals and their characteristics. The characteristics can be continually refined and added to because no other database parts depend on them; no other parts exist on a level lower than characteristics.

Implementing the Database for Simplicity

DMSII offers many features for implementing a database. However, you should use only those features that best suit your application needs.

A simple design is a safe design. It facilitates application programming, operating simplicity, and easier recovery. You can implement simple designs with the following guidelines:

- Use simple records.
- Use simple structures.

Using Simple Records

Use only one record for each individual element in a data set and include all the characteristics of that element in the same record.

If you split an element across two or more records, you may update one of the records of the element without updating the other records.

Using Simple Structures

Use disjoint structures, foreign keys, index sequential sets, and automatic subsets rather than embedded structures, links, and manual subsets.

Disjoint Data Sets

Disjoint data sets offer the following advantages over embedded data sets:

- Simple tables can represent disjoint data sets. Only more complex hierarchies can represent embedded structures.
- Automatic sets and subsets can be used in disjoint data sets and can be easily recovered during database failure.

  Because automatic sets and subsets derive all the information from the disjoint structure, except for the order of duplicates, they can be recovered easily and quickly using reorganization.

  Recovery and reorganization are explained in Section 6, "Restoring a DMSII Database."
- Because they stand on their own, disjoint data sets are easier to query and maintain than embedded data sets. Records in disjoint data sets can be accessed directly. To access records in embedded data sets, you must go through the master record before going to the embedded one.

Foreign Key Technique

The *foreign key technique* uses data items to associate records from two different data sets without using links. These data items are known as *foreign keys*. A foreign key copies a data item from another data set into the original data set so that items from the foreign data set can be found.

For example, to relate employee records and project records together, the EMPLOYEES data set can contain an extra data item called ASSIGNED-PROJECT. This data item duplicates the PROJ-NBR data item from the PROJECTS data set, as the following example shows:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    ASSIGNED-PROJECT      NUMBER(Ø4);
  );

ALLPROJECTS SET OF EMPLOYEES
  KEY IS ASSIGNED-PROJECT;

PROJECTS DATA SET
  (
    PROJ-NAME             ALPHA (15);
    PROJ-NBR              NUMBER(Ø4);
  );
```

The ALLPROJECTS set of the EMPLOYEES data set then uses the ASSIGNED-PROJECT data item as a foreign key for the PROJECTS data set. To find the projects assigned to an employee, you go through the ALLPROJECTS set and find the ASSIGNED-PROJECT of EMPLOYEES that matches the PROJ-NBR of PROJECTS.

Index Sequential Sets and Automatic Subsets

Index sequential sets and automatic subsets offer the following advantages over links and manual subsets:

- DMSII can automatically maintain index sequential sets and automatic subsets. Only application programs can maintain links and manual subsets.
- Index sequential sets automatically order records. Only application programs can order records in links.
- Programs call data management operations less frequently when the database they access uses index sequential sets and automatic subsets.

Turning Embedded Structures into Disjoint Structures

Simple structures can generally represent all the relationships that an embedded structure can. Even relationships that were originally embedded can often be turned into disjoint relationships.

For example, consider the following EMPLOYEES data set with an embedded DEPENDENTS data set:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
      DEPENDENTS ORDERED DATA SET
        (
          DEP-NAME        ALPHA (15);
          DEP-AGE         NUMBER(Ø3);
        );
  );

NAMESET SET OF EMPLOYEES
  KEY IS EMP-NAME;
```

You can turn this embedded relationship into a disjoint one by following these steps:

1. Remove the DEPENDENTS data set from the EMPLOYEES data set and turn DEPENDENTS into a disjoint data set.
2. Add a new data item called DEP-PARENTNAME to the DEPENDENTS data set that copies the value of EMP-NAME in the EMPLOYEES data set.
3. Add a set called DEPSET that uses DEP-PARENTNAME as its primary key.

   These steps produce the following two disjoint data sets:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    EMP-NBR               NUMBER(Ø6);
    EMP-ADDRESS           ALPHA (2Ø);
    EMP-PAY-TYPE          ALPHA (Ø1);
  );

NAMESET SET OF EMPLOYEES
  KEY IS EMP-NAME;

DEPSET SET OF DEPENDENTS
  KEY IS DEP-PARENTNAME;

DEPENDENTS ORDERED DATA SET
  (
    DEP-PARENTNAME        ALPHA (15);
    DEP-NAME              ALPHA (15);
    DEP-AGE               NUMBER(Ø3);
  );
```

Choosing Relationships

After defining the structures of a database, you can choose how to best define relationships between records in the database. There are three kinds of database relationships: one-to-one, one-to-many, or many-to-many.

The following paragraphs discuss examples of these relationships. All these examples show how to use simple structures to represent relationships in one data set or between two data sets.

One-to-One (1:1) Relationships

One-to-one (1:1) relationships exist when one record relates to exactly one other record, and vice versa. For example, assume that an employee can only have one project and a project only one employee as shown in Figure 4-3.

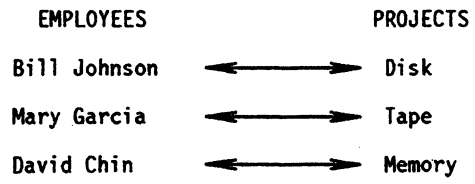

Figure 4–3. One-to-One (1:1) Relationships

One-to-One (1:1) Relationships within One Data Set

Assume a PEOPLE data set, in which a person can have a 1:1 relationship with another person through marriage. This relationship is shown by Figure 4-4.

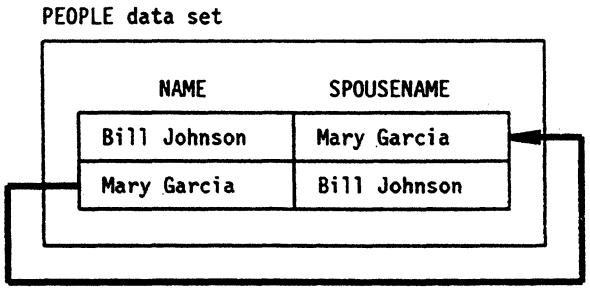

Figure 4–4. One-to-One (1:1) Relationships within One Data Set

This diagram is represented by the following declaration:

```
PEOPLE DATA SET
  (
    NAME                ALPHA (15);
    SPOUSENAME          ALPHA (15); % NULL if not married
  );

NAMESET SET OF PEOPLE
  KEY IS NAME;
```

To find the spouse of any person, you search NAMESET with the following condition:

```
NAME = SPOUSENAME
```

One-to-One (1:1) Relationships between Two Data Sets

Assume a 1:1 relationship between the EMPLOYEES and PROJECTS data sets. Each employee can have only one project, and each project can have only one employee as shown in Figure 4-5.

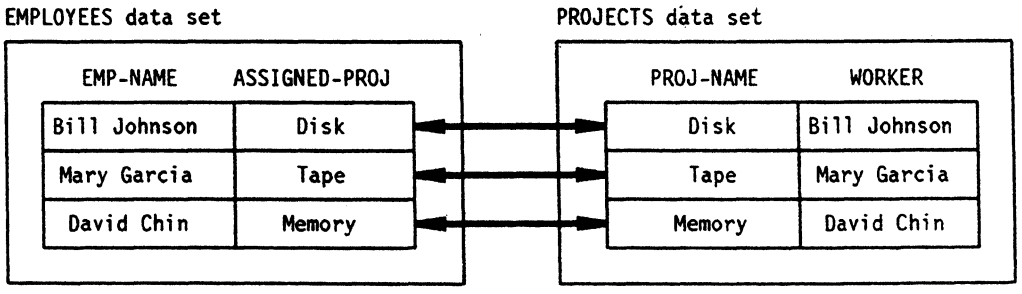

Figure 4–5. One-to-One (1:1) Relationships between Two Data Sets

This diagram is represented by the following declaration:

```
EMPLOYEES DATA SET
  (
    EMP-NAME                ALPHA (15);
    ASSIGNED-PROJ           ALPHA (20);
  );

EMPSET SET OF EMPLOYEES
  KEY IS EMP-NAME;

PROJECTS DATA SET
  (
    PROJ-NAME               ALPHA (20);
    WORKER                  ALPHA (15);
  );

PROJSET SET OF PROJECTS
  KEY IS PROJ-NAME;
```

Using an employee from the EMPLOYEES data set, you can find the related project in the PROJECTS data set by using PROJSET with the following condition:

```
ASSIGNED-PROJ = PROJ-NAME
```

Conversely, using a project from the PROJECTS data set, you can find the related employee in the PROJECTS data set by using EMPSET with the following condition:

```
WORKER = EMP-NAME
```

One-to-Many (1:M) Relationships

One-to-many (1:M) relationships exist when one record can relate to many records, but each of the many records can relate back to only one record. For example, assume a project can have many employees but each employee can work on only one project. This relationship is shown by Figure 4-6.

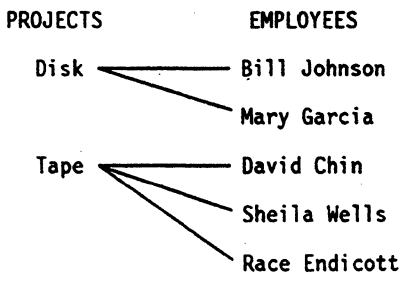

Figure 4–6. One-to-Many (1:M) Relationships

One-to-Many (1:M) Relationships within One Data Set

Assume that in the EMPLOYEES data set, a manager can have many subordinates, but each subordinate can have only one manager. Those relationships are shown by Figure 4-7.

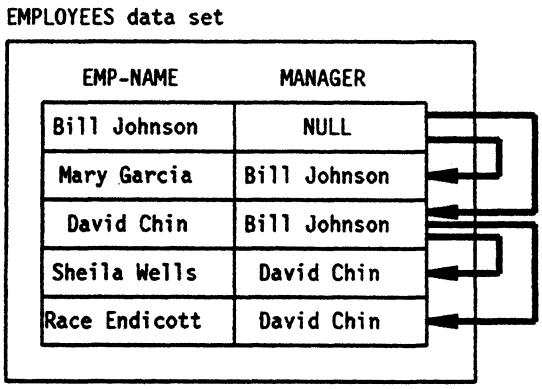

Figure 4–7. One-to-Many (1:M) Relationships within One Data Set

This diagram is represented by the following declaration:

```
EMPLOYEES DATA SET
  (
    EMP-NAME              ALPHA (15);
    MANAGER               ALPHA (15); % NULL if no manager
  );

EMPSET SET OF EMPLOYEES
  KEY IS EMP-NAME;
```

To find the manager of any person, you search EMPSET with the following condition:

```
EMP-NAME = MANAGER
```

To find the subordinates of a manager XNAME, you search EMPSET to find all persons who fulfill the condition:

```
MANAGER = "XNAME"
```

One-to-Many (1:M) Relationships between Two Data Sets

Assume 1:M relationships between the PROJECTS and EMPLOYEES data sets. Each project can have several employees but each employee can only have one project. These relationships are shown by Figure 4-8.

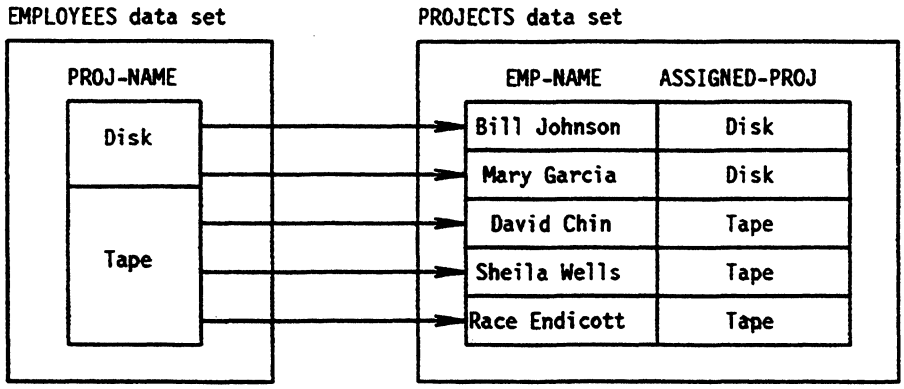

Figure 4–8. One-to-Many (1:M) Relationships between Two Data Sets

This diagram is represented by the following declaration:

```
PROJECTS DATA SET
  (
    PROJ-NAME              ALPHA (20);
  );

PROJSET SET OF PROJECTS
  KEY IS PROJ-NAME;

EMPLOYEES DATA SET
  (
    EMP-NAME               ALPHA (15);
    ASSIGNED-PROJ          ALPHA (20);
  );

EMPSET SET OF EMPLOYEES
  KEY IS EMP-NAME;
```

To find the projects of an employee, you search EMPSET with the following condition:

```
ASSIGNED-PROJ = PROJ-NAME
```

To find the workers on project XPROJ, you search PROJSET to find all persons who fulfill the condition:

```
ASSIGNED-PROJ = "XPROJ"
```

Many-to-Many (M:N) Relationships

Many-to-many (M:N) relationships exist when one record can relate to many records, and vice versa. For example, assume that a project can have many employees and each employee can work on many projects. These relationships are shown by Figure 4-9.

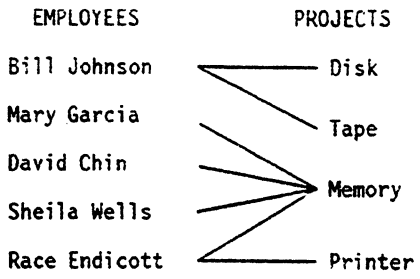

Figure 4–9. Many-to-Many (M:N) Relationships

Foreign keys cannot define M:N relationships. Although an embedded manual subset can, this subset is difficult to maintain and update. A simpler method is to use the relationship data set technique.

Relationship Data Set Technique

The *relationship data set technique* uses a disjoint data set to define a relationship. The relationship data set stores one record for each connection between a pair of records. Each of the two items in that record comes from each half of the paired records.

The relationship data set technique is useful whether you are defining relationships in one data set or between two data sets.

Relationship Data Set Technique and One Data Set

Assume that each member of the EMPLOYEES data set can be a manager or subordinate. A manager can have many subordinates, and each subordinate can have many managers. So, a M:N relationship exists between the members of the EMPLOYEES data set. This relationship is stored in a relationship data set called MGR-SUB.

These relationships are shown by Figure 4-10.

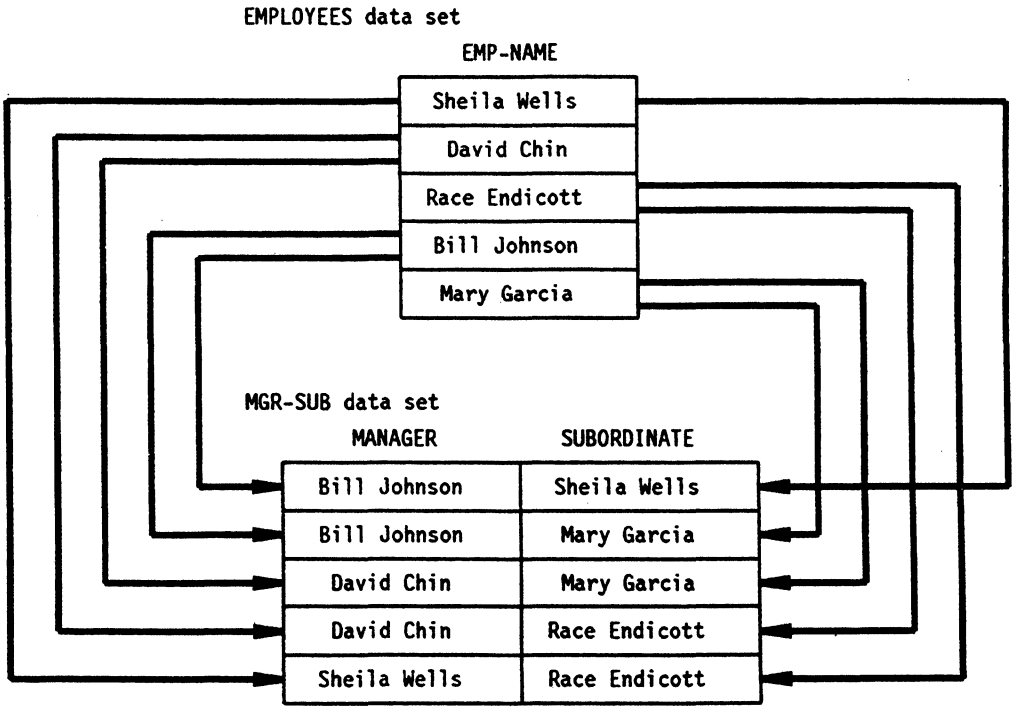

Figure 4–10. Relationship Data Set and One Data Set

This diagram is represented by the following declaration:

```
EMPLOYEES DATA SET
  (
    EMP-NAME                ALPHA (15);
  );

NAMESET SET OF EMPLOYEES
  KEY IS EMP-NAME;

MGR-SUB DATA SET
  (
    MANAGER                 ALPHA (15);
    SUBORDINATE             ALPHA (15);
  );

MGRTOSUB SET OF MGR-SUB
  KEY IS MANAGER;

SUBTOMGR SET OF MGR-SUB
  KEY IS SUBORDINATE;
```

Note that the MGR-SUB relationship data set has two automatic sets to ease retrieval. The keys of the two sets are the MANAGER data item and the SUBORDINATE data item.

To find the managers of subordinate XNAME, you search the SUBTOMGR set with the following condition:

```
SUBORDINATE = "XNAME"
```

Conversely, to find the subordinates of a manager YNAME, you search the MGRTOSUB set with the following condition:

```
MANAGER = "YNAME"
```

Relationship Data Set Technique and Two Data Sets

Assume that a project can have several employees, and each employee can have several projects. As a result, M:N relationships exist between the PROJECTS and EMPLOYEES data sets. These relationships are stored in a data set called WORK.

Figure 4-11 illustrates this situation.

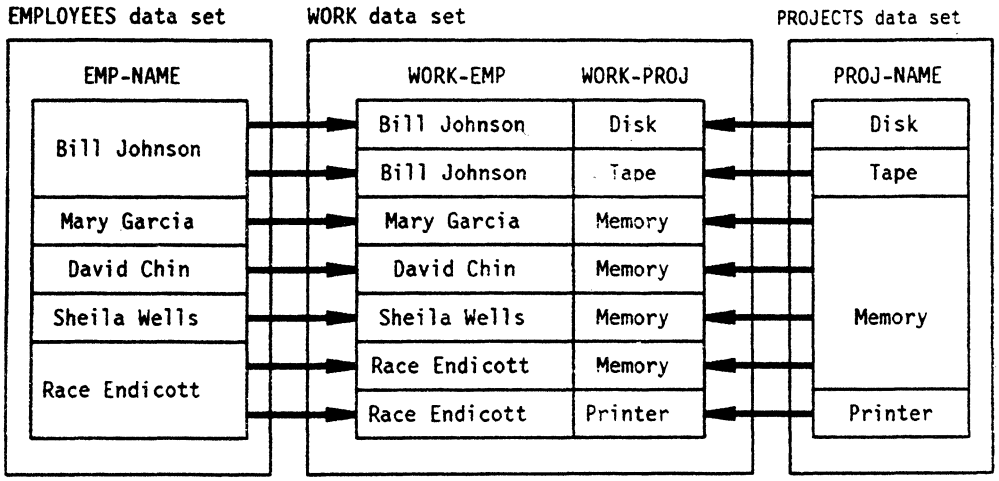

Figure 4–11. Relationship Data Set and Two Data Sets

This diagram is represented by the following declaration:

```
EMPLOYEES DATA SET
  (
    EMP-NAME                ALPHA (15);
  );

EMPSET SET OF EMPLOYEES
  KEY IS EMP-NAME;

PROJECTS DATA SET
  (
    PROJ-NAME               ALPHA (20);
  );

PROJSET SET OF PROJECTS
  KEY IS PROJ-NAME;

WORK DATA SET
  (
    WORK-EMP                ALPHA (15);
    WORK-PROJ               ALPHA (20);
  );

EMPTOPROJ SET OF WORK
  KEY IS WORK-EMP;

PROJTOEMP SET OF WORK
  KEY IS WORK-PROJ;
```

Note that the WORK relationship data set has two automatic sets to ease retrieval between the original data sets:

- The key of EMPTOPROJ is the unique WORK-EMP data item, which is a copy of the EMP-NAME data item of the EMPLOYEES data set.
- The key of PROJTOEMP is the unique WORK-PROJ data item, which is a copy of the PROJ-NAME data item of the PROJECTS data set.

To find the projects of an employee XNAME you search the EMPTOPROJ set with the following condition:

```
WORK-EMP = "XNAME"
```

To find the workers on project XPROJ, you search the PROJTOEMP set with the following condition:

```
WORK-PROJ = "XPROJ"
```

Defining a Database to the System

When you are finished defining your database on paper, you can define your database to DMSII. You do this in one of two ways:

- By writing a Data and Structure Definition Language (DASDL) source file
- By using Advanced Data Dictionary System (ADDS) menus

Writing a DASDL Source File

You can use DASDL to define your database by writing DASDL statements in a source file. When the DASDL source is compiled, it creates or updates the database description file.

For more information on DASDL, see the DMSII DASDL Reference Manual.

Using ADDS Menus

You can define your database using ADDS instead of DASDL. ADDS provides centralized definition, storage, and retrieval of DMSII data definitions, as well as conventional files. Through ADDS, you can define and manage a database interactively by filling out menus.

For more information on ADDS, see the InfoExec Administration Guide.

Compiling Tailored Software

The next step in creating a database is to compile tailored software. Tailored software is software that is specific to one database. Figure 4-12 shows how tailored software is compiled.

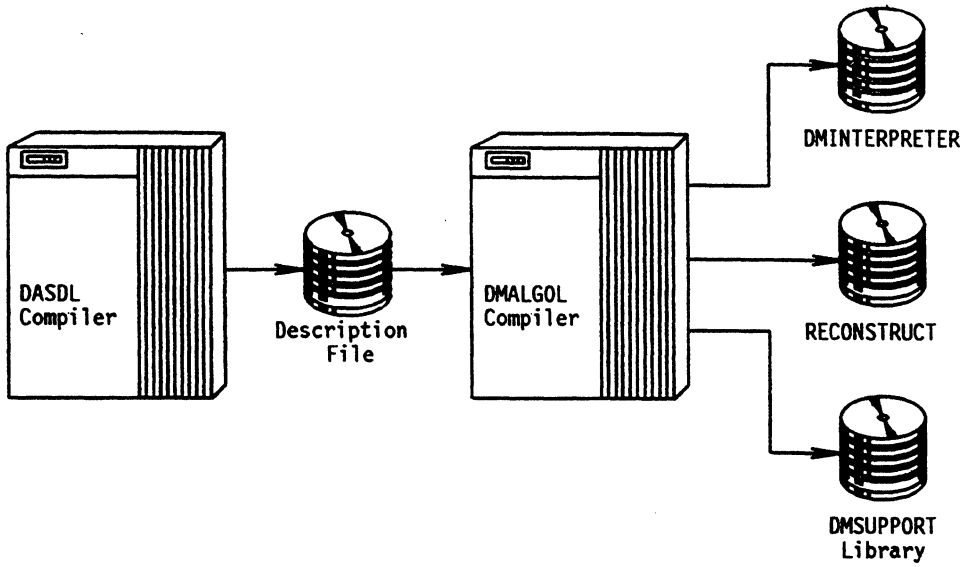

Figure 4–12. Compiling Tailored Software

The DASDL compiler produces the description file. The DMALGOL compiler uses the description file to create the DMSUPPORT library, DMINTERPRETER, and the RECONSTRUCT program.

The DMSUPPORT library is described in Section 1, "Introducing the Data Management System II (DMSII)." DMINTERPRETER is described in Section 7, "Accessing a DMSII Database." The RECONSTRUCT program is described in Section 6, " Restoring a DMSII Database."

Initializing a Database

Before you can use a database, you must initialize both the control file and the database files. Initialization creates the actual files.

To initialize the control file, use SYSTEM/DMCONTROL. The initialized control file ensures that the software and the database are compatible and that all the database files are compatible. The control file also stores audit control information and other database parameters.

To initialize the database files, use the INITIALIZE statement of the DMUTILITY program.

Running SYSTEM/DMCONTROL and SYSTEM/DMUTILITY are described in the DMSII Utilities Operations Guide.

Section 5
Changing a DMSII Database

During the life of a database, the demands on it might change. Three processes can change an existing database description: update, reorganization, and modeling.

When the DASDL compiler creates a new database description, it sets a number stored in the description file to 1. This number is the update level of the database.

If the update level of an application program is not compatible with the update level of the database it is running against, that program cannot run against that database.

Updating a Database

When updating a database, you can change several options, parameters, and physical options; add and delete disjoint data sets, sets, and subsets; and change existing file formats.

For more information on updating, refer to the DMSII DASDL Reference Manual and the DMSII Utilities Operations Guide.

Reorganizing a Database

Reorganization changes the database update level because it changes the format of database files and records. The two forms of reorganization, format conversion and garbage collection, are explained in the following paragraphs.

For more information on reorganizing, refer to the DMSII DASDL Reference Manual and the DMSII Utilities Operations Guide.

Format Conversion

Format conversion reorganizes database formats on two levels:

- File format conversion

  When you change the physical options of a database, you change file formats. Such options include the AREASIZE or BLOCKSIZE options for a data set, the MODULUS option for an Access, or the TABLESIZE option for a subset.

- Record format conversion

  When you add or delete data items, you change record formats. This form of reorganization changes the format of data sets or global data. If any programs directly accessed the added or deleted data items, recompile those programs. You do not have to recompile programs that use remaps of the added or deleted data items if the remaps are unchanged.

Garbage Collection

Garbage collection cleans up a database by

- Consolidating unused space in data sets, sets, and subsets, and returning this space to the system
- Reordering data sets, sets, and subsets to minimize access time
- Generating a new set or automatic subset from an existing data set

After you perform one of the types of reorganization, you run a program called SYSTEM/BUILDREORG. This program creates the REORGANIZATION program, which generates the new data sets, sets, or subsets, and removes the old ones.

Modeling a Database

You can create a model of an existing database to be used for testing or production. Both the model and the original share the same DMSUPPORT library. Both also have the same structures and formats. Modeling does not change the update level of the database.

Changing the database to include the MODEL option creates the model when the database is compiled and initialized. At compilation, you can also change certain options for the model, such as the guard file titles, AREAS, and REBLOCKFACTOR.

Application programs can run interchangeably against the model or the existing database and need not be recompiled.

For more information on modeling, refer to the DMSII DASDL Reference Manual.

Section 6
Restoring a DMSII Database

Sometimes, database processing is interrupted because the system crashes, a processing error happens, or a program is aborted. Such interruptions can corrupt data and make the database unusable. DMSII provides three ways of ensuring that your database can be restored to its correct state:

- Backing up the database. The backups can then be reloaded.
- Auditing the database by storing its *updates*. An update is any information that is added, deleted, or changed in a database. These updates can then be reapplied.
- Using synchronized recovery to record the messages that produced the database changes. These messages can then be sent again.

This section discusses these methods.

Backing Up a Database

The easiest way to restore a database is to load a copy of it from a tape. The process of backing up a database to tape is called *dumping*.

You dump a database through the DMUTILITY program. DMUTILITY is further described in the DMSII Utilities Operations Guide.

Dumping a Database

You dump a database using the DMUTILITY *DUMP* statement. DMUTILITY verifies the timestamps of the structures and the database using the control file.

There are two kinds of dumps:

- Offline dump

  DMUTILITY waits for all update programs to finish processing the database before dumping. Then, DMUTILITY locks the control file, preventing updates to the database. Inquiry programs can continue to access the database. After a successful dump, DMUTILITY unlocks the control file for updating. This is a safer type of dump than an online dump.

- Online dump

  Online dumps work only on audited databases because such dumps are used with the audit process. This process is explained later in this section.

Maintaining a Dump Tape Directory

The dump tape directory records information about DMUTILITY database dumps. It consists of two parts, each of which is stored, by default, on the same pack as the control file.

- Main directory

  Each database has one main directory called < database name > /DMDUMPDIR. The main directory records an entry for each dump since the directory was enabled.

  SYSTEM/DMDUMPDIR creates the main directory.

- Dump directory

  Each entry in the main directory produces a dump directory called

  ```
  <database name>/DMDUMPDIR/<dump time>/<dump name>
  ```

  The name of the dump directory describes the time the database was dumped and the name of the dump tape.

  Each dump directory contains the name of the dump tape and a list of the database files dumped to the tape.

  The BUILDDUMPDIRECTORY statement of the DMUTILITY program creates the dump directory and inserts dump directory entries in the main directory.

For more information on SYSTEM/DMDUMPDIR and the BUILDDUMPDIRECTORY statement, see the DMSII Utilities Operations Guide.

Auditing a Database

Copying a backup dump is a fairly simple way of restoring a database. However, if the database was updated after you did the backup, the dump tape will not contain those updates. To restore the database to its updated state, you have to load the database from tape and perform the missing updates again.

DMSII provides you with the ability to *audit* your database. Auditing creates a log of all database activity. This log is called an *audit trail*, and consists of one or more physical files stored on disk or tape. These files are used by the recovery software to restore the database to a usable state.

By reapplying the updates from this audit trail, recovery can quickly restore the audited database. To create an audited database, use the AUDIT option at the beginning of a DASDL declaration:

```
OPTIONS (AUDIT);

RESTARTAREA RESTART DATA SET
 (
   RDS-TRAN-COUNT        TRANSACTION COUNT;
   RDS-RECD-TYPE         RESTART TYPE;
   RDS-USER-KEY          ALPHA (18);
   RDS-USER-INFO         ALPHA (200);
 );

EMPLOYEES DATA SET
 (
   NAME                  ALPHA (12);
 );
       .
       .
```

Recording a Transaction

Any operations that update the database must be entered between the BEGINTRANSACTION and ENDTRANSACTION statements as follows:

```
BEGINTRANSACTION
    <change to the database>
ENDTRANSACTION
```

These statements identify a *transaction*, which is a single, logical change to the database. While the program operates between these statements, it is in *transaction state*.

For example, assume your system processes the following COBOL74 program fragment. (RESTART-AREA is the name of the restart data set. Restart data sets are described in this section under "Restart Data Set.")

```
128100      LOCK NEXT EMPLOYEES.
128200      COMPUTE EMP-HOURLY-RATE = EMP-HOURLY-RATE * 1.10.
128300      BEGIN-TRANSACTION RESTART-AREA.
128400      STORE EMPLOYEE.
128500      END-TRANSACTION RESTART-AREA.
```

The transaction produces the change in the audit trail shown in Figure 6-1.

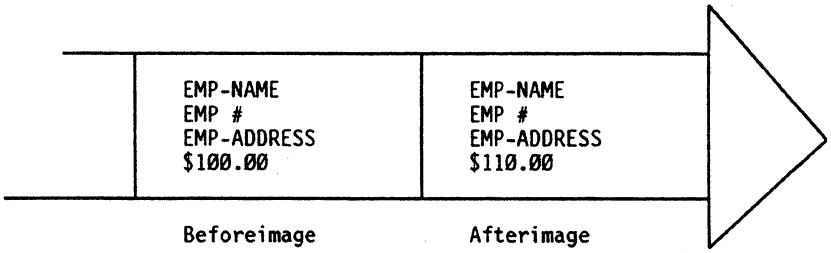

Figure 6–1. Audit Trail

As Figure 6-1 illustrates, the audit trail registers the change made to the record as a pair of images. The *beforeimage* contains the original salary. The *afterimage* contains the original salary multiplied by 1.1.

The beforeimage is an image of the database record before the update. If a record is deleted, only its beforeimage exists in the audit trail. Applying a beforeimage to restore the database is also known as *backing out a transaction*.

The afterimage is an image of the database record after the change. If a record is added, only its afterimage exists in the audit trail. Applying an afterimage to restore the database is also known as *reapplying a transaction*.

The following statements are allowed only in transaction state: STORE, DELETE, GENERATE, INSERT, REMOVE, and ASSIGN.

For information on programming to maximize the use of transactions and the audit trail, see the DMSII Application Programming Guide and the appropriate language reference manual.

Designating Audit Trail Attributes

When you declare an audited database, you can designate the following attributes to control the audit trail:

- ALTERNATE

  Switches the primary and secondary audit from pack to tape when there is not enough space on pack.

- DUPLICATED

  Designates a second copy of the audit trail that is maintained automatically by the Accessroutines. If recovery finds an error on the original audit trail, it automatically switches to the copy.

- KIND

  Designates whether the audit file is on disk or tape or both.

- UPDATE EOF

Designates how often the pointer to the end of the audit trail is updated. This option is useful only if the audit file is on pack or disk.

- VERIFY

  Verifies the audit file with consistency checks.

Other audit trail attributes include AREAS, AREASIZE, BLOCKSIZE, CHECKSUM, and SECURITYGUARD, which are described under "Physical Options" in Section 2, "Understanding DMSII Database Concepts."

The following example declares an area size of 100 blocks, the number of areas as 88, and an end-of-file (EOF) pointer update frequency of once every 50 blocks.

```
AUDIT TRAIL
  (
    AREAS = 88,
    AREASIZE = 1ØØ BLOCKS,
    UPDATE EOF = 5Ø BLOCKS
  );
```

Marking Quiet Points, Syncpoints, and Controlpoints

To minimize recovery time and to ensure that recovery processes only completed transactions, DMSII marks certain points in time with *quiet points*, *syncpoints*, and *controlpoints*. These points are described in the following text.

Quiet Points

A quiet point is a time when a database and all its update programs are not in transaction state. Figure 6-2 shows a quiet point. (BTR and ETR stand for the BEGINTRANSACTION and ENDTRANSACTION statements.)

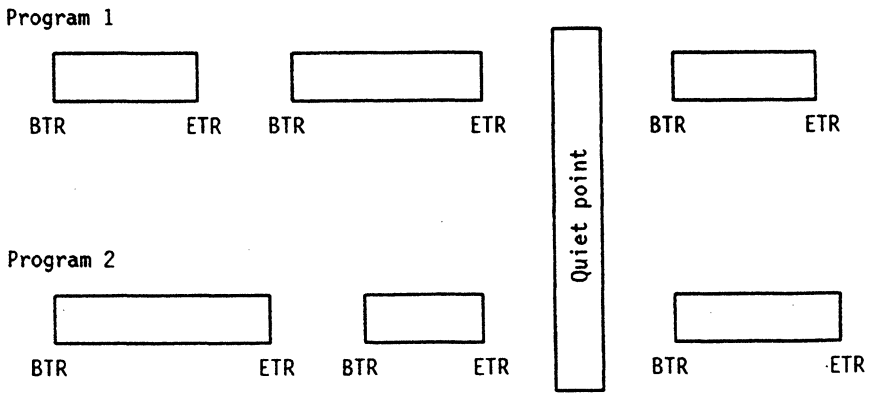

Figure 6–2. Quiet Point

In Figure 6-2, the quiet point happens after the second transaction in both the first and second program.

When a transaction fails, recovery backs out all transactions until a quiet point is reached. *Backing out* reverses changes made to a database and rolls back each transaction to a consistent state at an earlier point in time.

Syncpoints

If too many programs update a database, that database may never go out of transaction state. DMSII cannot create a quiet point unless you force a quiet point after a certain number of transactions. This forced quiet point is known as a syncpoint. You control the number of transactions that happen before each syncpoint with the SYNCPOINT parameter. Figure 6-3 shows a syncpoint.

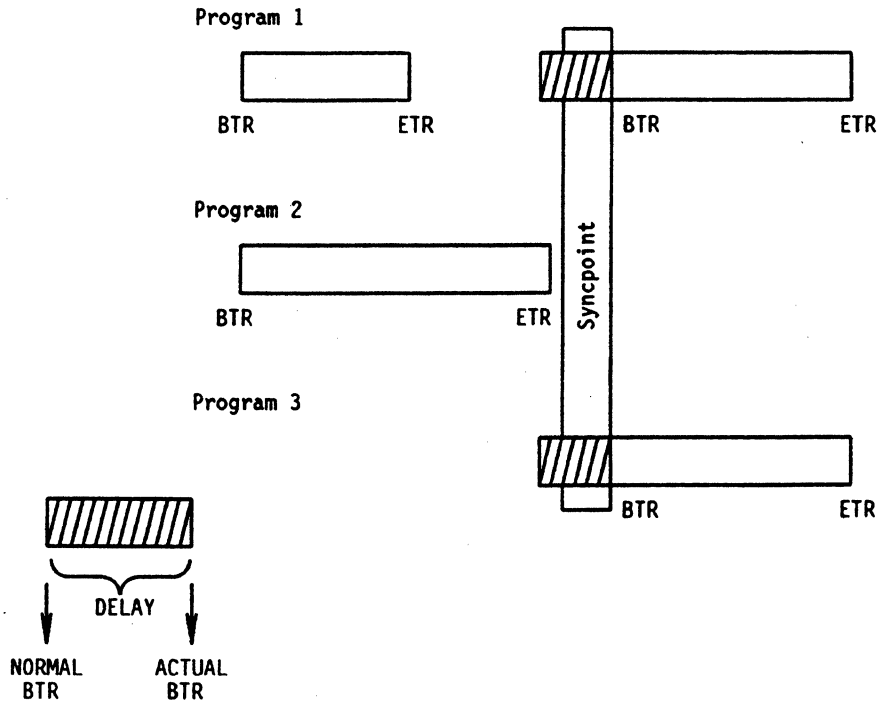

Figure 6–3. Syncpoint

In this case, a syncpoint prevents programs 1 and 3 from beginning their transactions until program 2 has ended its transaction.

Another way to create a syncpoint is to use the DASDL option REAPPLYCOMPLETED, which forces an audit after each ENDTRANSACTION statement. When using the concurrency control options during a forced audit, other programs can be in transaction state, whereas during a syncpoint, no program can be in transaction state.

You can also control the amount of time a program waits for a syncpoint with the SYNCWAIT parameter. This parameter requires that you set the DASDL options REAPPLYCOMPLETED and INDEPENDENTTRANS.

For more information on SYNCPOINT, REAPPLYCOMPLETED, INDEPENDENTTRANS, and SYNCWAIT, see the DMSII DASDL Reference Manual. The INDEPENDENTTRANS and REAPPLYCOMPLETED options are also explained under "Recovering a Database" in this section.

Controlpoints

The Accessroutines stores changed records in buffers until it can conveniently write them to the database files. The point at which those records are written to the files is called a controlpoint. By default, these control points happen after every two syncpoints. However, you can designate the number of syncpoints between controlpoints with the CONTROLPOINT parameter.

For more information on the CONTROLPOINT parameter, see the DMSII DASDL Reference Manual.

Examining and Copying Audit Files

To examine audit files, use SYSTEM/PRINTAUDIT. With this utility, you can print or display part or all of an audit file. You can do so either interactively or through statements in a job.

To free up space on your pack, use SYSTEM/COPYAUDIT. With this utility, you can copy audit files from one medium to another and verify them.

Both programs are described in the DMSII Utilities Operations Guide.

Declaring the Restart Data Set

In a recovery situation, programs decide which transactions to reapply by looking at information in the restart data set. Restart data sets identify the programs that access a database and the points at which those programs start after a recovery. Each audited database must have one restart data set.

The following example declares a restart data set called RESTARTAREA:

```
RESTARTAREA RESTART DATA SET
  (
    RDS-TRAN-COUNT          TRANSACTION COUNT;
    RDS-RECD-TYPE           RESTART TYPE;
    RDS-USER-KEY            ALPHA (18);
    RDS-USER-INFO           ALPHA (200);
  );
```

The first two items, RDS-TRAN-COUNT and RDS-RECD-TYPE, are allocated whether or not you declare them. However, to use these records in a program, you need to declare them. RDS-TRAN-COUNT counts the number of times the program was in transaction state. RDS-RECD-TYPE identifies the source of the restart record.

The user program stores information in the following items:

- RDS-USER-KEY

  Stores the name of a program that was working on the database during recovery

- RDS-USER-INFO

  Stores information that tells the program where to restart

The restart data set is explained further in the DMSII DASDL Reference Manual.

Recovering a Database

During system failure, the system normally recovers the database automatically. However, in some cases—for example, during pack failure—you need to recover the database manually. Both cases are described in the following paragraphs.

Recovering Automatically

There are three ways in which the system automatically recovers data:

- Abort recovery

  Backs out transactions.

- Halt/load recovery

  Reapplies transactions, returns the database to the point before the halt/load, and then backs out any partial changes.

- COMS synchronized recovery

  Used when recovering from an abort or halt/load. Database transactions are reapplied first, then Communications Management System (COMS) transactions are reapplied.

These types of automatic recovery, and the concurrency control features of DMSII are explained in the following paragraphs.

Abort Recovery

A program or process is aborted when an operator or a program uses the *?DS (Discontinue)* system command, or during program errors such as closing the database in transaction state. For more information on this command, see the System Commands Reference Manual.

Abort recovery backs out all transactions until it reaches a quiet point. As a result, any incomplete transactions that might have corrupted the database are undone, and the database returned to a correct state.

Halt/Load Recovery

The system performs a halt/load under the following circumstances:

- To initialize system software
- To recover from an error that the operating system cannot handle

When the database is first opened after a halt/load, DMSII automatically runs halt/load recovery. At this point, the Accessroutines prevents the database from being opened, forcing all programs and users to wait until recovery finishes.

Although halt/load recovery is automatic, you can run SYSTEM/DMRECOVERY manually as described in the DMSII Utilities Operations Guide. If recovery is not needed or is already running, your recovery run stops with an appropriate message.

Concurrency Control

The purpose of concurrency control is to ensure that all completed transactions are recovered after an abort or halt/load. DMSII supplies you with two DASDL options (INDEPENDENTTRANS and REAPPLYCOMPLETED) that provide you with concurrency control.

INDEPENDENTTRANS

The DASDL *INDEPENDENTTRANS* option creates independent transactions to ensure your transactions do not affect those of other users. Recovery can then back out all or part of the aborted transaction without affecting other users.

Although INDEPENDENTTRANS is useful for both abort recovery and halt/load recovery, you can use it in coordination with the following language extensions to optimize abort recovery:

- DMSAVEPOINT

  Provides an intermediate transaction point record for audit. The transaction points apply only to the current transaction, and do not affect halt/load recovery.

- ABORTTRANSACTION

  Discards any updates made in a transaction after a BEGINTRANSACTION. Removes a program from transaction state.

Refer to the DMSII DASDL Reference Manual for instructions on setting the INDEPENDENTTRANS option. Refer to the DMSII Application Programming Guide and the appropriate language manual for information on DMSAVEPOINT, ABORTTRANSACTION, and other language extensions.

REAPPLYCOMPLETED

The DASDL *REAPPLYCOMPLETED* option is used with the INDEPENDENTTRANS option and causes all completed transactions to be reapplied from the audit trail after a system halt/load. Refer to the DMSII DASDL Reference Manual for instructions on setting the REAPPLYCOMPLETED option.

COMS Synchronized Recovery

The Communications Management System (COMS) synchronized recovery ensures that no complete transactions are lost during a recovery from a program abort or system halt/load.

In abort recovery and halt/load recovery, the changes to the database are stored by DMSII in an audit trail. After a system failure, DMSII uses the audit trail to restore the database.

In synchronized recovery, the messages to the database program are stored by COMS in a similar log called a *transaction trail*.

If the database software informs COMS that some transactions were not recovered using the audit trail, COMS uses the transaction trail to resubmit the transactions to the database applications in the order in which the transactions were originally entered.

Usually, if you use the INDEPENDENTTRANS option and the REAPPLYCOMPLETED option, synchronized recovery will not be used.

Figure 6-4 shows how a message goes from a terminal to the database file, and at what point the message is stored in the transaction trail and audit trail.

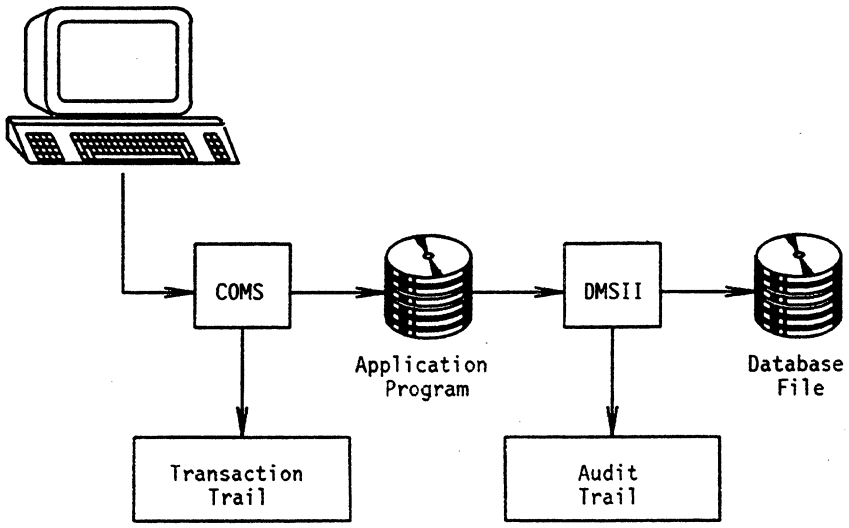

Figure 6–4. Transaction Trail and Audit Trail Storage

For example, a transaction trail keeps track of the messages sent by an agent to the database program to change a salary, while an audit trail keeps track of a salary as it existed before and after a raise.

During normal processing, user programs add an additional statement called MIDTRANSACTION. This statement tells the message control system (MCS) to record certain values in the restart data set record. The MIDTRANSACTION point is a marker

showing that the transaction has locked all of its resources. COMS replays transactions based on the order in which they reached MIDTRANSACTION.

For more information, see the COMS Operations Guide.

Recovering Manually

You can also recover the database manually using the DMUTILITY program. This program, along with the DMRECOVERY, DMDATARECOVERY, and RECONSTRUCT programs, performs the following kinds of recovery:

- Partial database recovery, to
  - Recover one or more damaged rows of database files.
  - Recover one or more damaged structures.
  - Recover write errors by the *quickfix* process. This process recovers locked-out rows using the audit trail only; the rows do not have to be loaded from a backup dump.
- Whole database recovery, to
  - Rebuild the entire database, including recovery from data corruption.
  - Roll back the database to a point in time.
  - Recover unaudited databases using the COPY statement.

These types of recovery are explained in the following pages.

Partial Database Recovery

When you recover parts of the database, you are recovering damaged rows. So, partial database recovery is also known as *row recovery*. There are two forms of row recovery:

- Row reconstruction

  This type of recovery fixes read and write errors and pack crashes using audit files stored on tape. As a result, row reconstruction can restore a file that was removed from disk.

  Row reconstruction uses the tailored RECONSTRUCT program, which is called RECONSTRUCT/<database name>, and the data recovery program, which is called SYSTEM/DMDATARECOVERY.

- Quickfix process

  While row reconstruction can repair both read and write errors, the quickfix process repairs write errors only. Because quickfix uses the audit trail only, it is faster than normal row recovery. However, quickfix cannot always reconstruct all rows.

Use DMUTILITY and the appropriate parameters to begin row reconstruction.

Whole Database Recovery

To recover the entire database, the database must not be in use. Recovery is initiated by DMUTILITY and completed by DMRECOVERY. SYSTEM/DMRECOVERY is automatically initiated by DMUTILITY, unless NOZIP is specified in the RECOVER statement.

There are two methods for recovering the entire database.

- Rebuild

  This process moves the entire database forward in time. DMUTILITY begins rebuild by loading the entire database from one or more sets of dump tapes. Recovery then applies the audit trail afterimages to bring the database forward.

  Rebuild is most useful when the audit trail needed for halt/load recovery or row reconstruction is corrupted, or if an application program incorrectly processed the database.

  If the rebuild process is unsuccessful, you can still recover the database by reloading a backup copy of the database and manually redoing all the updates to the database.

- Rollback

  This process moves the entire database backward in time. It begins with the current database files. Recovery then applies the audit trail beforeimages to move the database back to a designated point in time.

  Rollback can save time over rebuild when you are correcting logical errors such as an update program that was run with wrong input. However, rollback cannot recover corrupted data.

With either method, recovery places a *discontinuity* into the current audit file to show that the database was rolled back or rebuilt. A discontinuity is a change made to a structure so that the structure no longer matches its audit image.

Section 7
Accessing a DMSII Database

After you have created and implemented your database, you can access it using the following:

- Extended Retrieval with Graphic Output (ERGO)

  An interactive program that produces reports

- Application program interfaces

  Application programs that access the database through either of the following:

  - Language extensions

    Programming language extensions that access databases at compile time.

  - Interpretive interface

    A run-time interface that accesses databases that have not been compiled. This interface enables applications that do not have DMSII extensions to directly access a database.

Figure 7-1 shows the ways you can access a DMSII database.

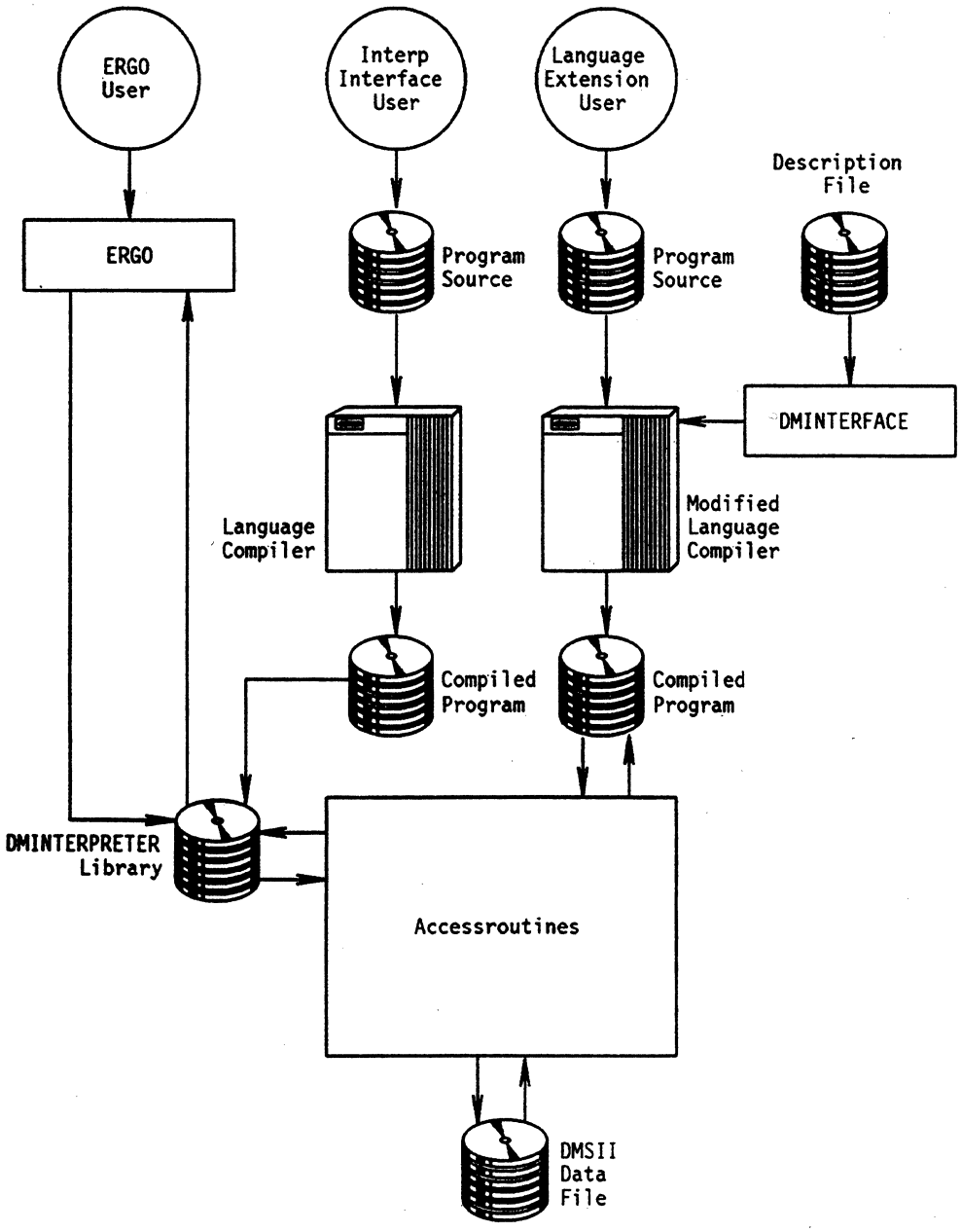

Figure 7–1. Accessing a DMSII Database

All three methods use the Accessroutines to access the database files. Both ERGO and the interpretive interface also use the DMINTERPRETER library.

Extended Retrieval with Graphic Output (ERGO)

ERGO is a program that retrieves data from a DMSII database to produce reports, display information graphically, and relate and format data. At the option of your site management, you can also use ERGO to change, add, or delete records in a database.

ERGO can manipulate data through built-in mathematical and statistical functions. Its reports includes tabular and statistical formats, and it can write information to both flat files and workstations in a variety of graphic formats: discrete and continuous plot graphs, bar graphs, kiviat diagrams, and histograms. You can use the standard report formats included in ERGO or define your own formats by controlling over 50 variables.

ERGO is designed for both inexperienced and experienced users. A menu-assisted interface prompts you through the processing of a request. In addition, help information is available for any command or data item. Experienced users can bypass the menus by entering direct commands.

For more information on ERGO, see the ERGO Operations Guide.

Application Program Interfaces

Application programs can access DMSII databases through two interfaces: a compile-time interface for compiled databases (language extensions) and a run-time interface for databases that have not been compiled (interpretive interface).

These interfaces are described briefly in the following text. For more information, see the DMSII Application Programming Guide and the appropriate language manual.

Language Extensions

This interface uses language extensions that require a valid database description file before an application can be compiled. These extensions appear in the following languages: ALGOL, COBOL, COBOL74, COBOL85 and RPG. This interface, called SYSTEM/DMINTERFACE, verifies that the language statements are correct.

Interpretive Interface

The interpretive interface provides direct access to DMSII databases through application languages that do not have DMSII extensions. It is a dynamic, run-time interface that allows an application program to be compiled before the databases it uses are created. This is possible because an application program never directly accesses the database but instead calls the DMINTERPRETER library, which is tailored to particular databases when the library is compiled.

Because you designate the DMINTERPRETER entry points after you run the application program, changes to the database do not normally require changes to the application library. Programs written in ALGOL, COBOL, COBOL74, COBOL85, or FORTRAN77 can use the interpretive interface.

Section 8
Maintaining a DMSII Database

DMSII provides the following programs for maintaining a database:

- Operations Control Manager (OCM)

  Uses menus to perform database maintenance
- DBANALYZER

  Provides reports based on analyses of the database structure
- DBCERTIFICATION

  Determines the integrity of the database
- DMMONITOR

  Interactively provides database and statistics information
- DMUTILITY

  Controls the operational aspects of a database
- VISIBLE DBS

  System commands that control system resources and communicate directly with the database
- LOADDUMP

  Generates programs that load a data set from a conventional file or dump a data set to a conventional file.

DMUTILITY, LOADDUMP, and the VISIBLE DBS come with the DMSII package. The other programs are optional.

These programs are explained further in this section.

Operations Control Manager (OCM)

With the Operations Control Manager (OCM), you can perform many tasks against the physical structure of your data. OCM manages database operations and control, providing for online control of such things as database dumps, database recovery, and database reorganization.

OCM offers the following advantages:

- With OCM, the database administrator or operations staff do not have to remember syntax. The menus guide them through long, complex commands without referring them to reference manuals. For experienced users who wish to bypass these menus, OCM also offers commands for performing database operations.
- OCM offers both online and batch operation. You can initiate a task online for immediate execution, or specify that the task be saved and executed later by a Work Flow Language (WFL) program generated by OCM.
- Online help screens assist you in any area concerning database operation and control.

To use OCM, the DMSII utilities must be on your system.

For more information about OCM, see the InfoExec Operations Guide.

DBANALYZER

DBANALYZER is a software tool that analyzes the logical and physical structure of a database and provides reports that a database manager can use to change, tune, reorganize, and document the database.

DBANALYZER describes the structures in the database and their relationships, analyzes the database files, and provides information about how the files are accessed.

You can use DBANALYZER interactively or in batch mode. Simple commands control each analysis, enabling you to direct the output to a terminal or a printer.

For more information about DBANALYZER, see the DMSII Utilities Operations Guide.

DBCERTIFICATION

The DBCERTIFICATION utility program determines the integrity of a DMSII database. It provides three levels of certification:

- Physical integrity

  The utility ensures that a file is physically intact and accessible, and isolates any problems at the data block level.

- Internal, intrafile integrity

  The utility verifies that data in a single structure are consistent, and that storage control and internal control information are correct.

- Interstructure integrity

  The utility ensures that relationships between data structures are correct.

This utility can be used interactively or in batch mode. You can select the structures in the database to verify and the tests to perform at each of the three levels.

For more information about DBCERTIFICATION, see the DMSII Utilities Operations Guide.

DMMONITOR

DMMONITOR is an interactive program that provides database status and statistics information and permits changes to database options and parameters. In addition, this program can sample database information and capture the information in a disk file for later analysis.

For more information about DMMONITOR, see the DMSII Utilities Operations Guide.

DMUTILITY

DMUTILITY is a program that controls the operational aspects of a database. With this program, you can

- Dump and load database files.
- Print data and control information for database files.
- Initialize all or specific structures in a database.
- Initialize nonautomatic forms of recovery for audited databases.

For more information, see the DMSII Utilities Operations Guide .

LOADDUMP

The LOADDUMP program generates programs that load a data set from a conventional file or dump a data set to a conventional file. The LOADDUMP program automatically generates and compiles BDMSCOBOL74 programs to perform the load or dump.

For more information about LOADDUMP, see the DMSII Utilities Operations Guide.

VISIBLE DBS

The VISIBLE DBS consists of system commands through which you can communicate directly with the database and control system resources as follows:

- Display database status, including the values of ALLOWEDCORE, the amount of buffer core in use, and audit file information.
- Change the values of ALLOWEDCORE, SHAREDALLOWEDCORE, LOCALALLOWEDCORE, SYNCPOINT, OVERLAYGOAL, and CONTROLPOINT.
- Switch audit files for audited databases.
- Print current statistics.
- Display status of individual structures, including the number of random and serial users, REBLOCK setting, buffer specifications, and REBLOCKFACTOR values.
- Determine what tasks are in transaction state and what tasks are waiting for locked records or a syncpoint.

For more information on the VISIBLE DBS, see the DMSII Utilities Operations Guide.

Glossary

A

abort
To terminate an active program or session abnormally and, sometimes, to attempt to restart it.

abort recovery
See recovery.

ABSN
See Audit Block Serial Number.

access
To perform an action on an object. Possible actions depend on the type of object; for example, interrogating or assigning a value to a variable, reading from or writing to a file, or invoking a procedure.

Access
A logical index structure that defines the physical ordering of records in direct, ordered, and random data sets. An access functions like a set, but no physical file is associated with an access.

Accessroutines
Routines that perform all physical and logical management of a database and allow many users to access the database concurrently. Each data management statement that a user language program executes invokes a portion of the Accessroutines to perform all file management functions that the statement requires.

ADDS
See Advanced Data Dictionary System.

Advanced Data Dictionary System (ADDS)
A software product that allows for the centralized definition, storage, and retrieval of data descriptions.

afterimage
The image of a database record in the audit trail after an update operation is performed.

ALGOL
Algorithmic language. A structured, high-level programming language that provides the basis for the stack architecture of the Unisys A Series systems. ALGOL was the first block-structured language developed in the 1960s and served as a basis for such languages as Pascal and Ada. It is still used extensively on A Series systems, primarily for systems programming.

alpha item
A data item that stores alphanumeric information (letters, numbers, special characters, and blanks) as EBCDIC characters. An alpha item cannot be used in calculations.

Audit Block Serial Number (ABSN)
A number associated with a block in the audit file. The first block in the first audit file is noted as ABSN 0. The ABSN is incremented by one through the last audit file.

audit file
A file produced by the Accessroutines that contains various control information, including beforeimages and afterimages of records resulting from changes to the database.

audit trail
A file produced by the Accessroutines that contains various control information, including beforeimages and afterimages of records resulting from change to the database. The audit trail is used to recover the database and supply restart information to programs after a hardware or software failure has occurred.

automatic subset
A subset declared with a condition that specifies which members of the data set are to be included in the subset. Entries are automatically inserted into or removed from the subset when records are added to or deleted from the data set.

B

back out
To undo changes made against a database and to roll back the progress of one or more transactions to a previously consistent state.

beforeimage
The image of a database record in the audit trail before an update operation is performed.

bit vector structure
A structure that enables unkeyed access to records of a data set. Each bit has an implicit relationship by position with one data set record.

Boolean item
A data item that stores information having a logical value of TRUE or FALSE.

buffer
An area in which data is stored temporarily.

C

CANDE
See Command and Edit.

COBOL

Common Business-Oriented Language. A widely used, procedure-oriented language intended for use in solving problems in business data processing. The main characteristics of COBOL are the easy readability of programs and a considerable degree of machine independence. COBOL is the most widely used procedure-oriented language.

COBOL74

A version of the COBOL language that is compatible with the ANSI X3.23-1974 standard.

Command and Edit (CANDE)

A time-sharing message control system (MCS) that enables a user to create and edit files, and develop, test, and execute programs, interactively.

Communications Management System (COMS)

A general message control system (MCS) that controls online environments on A Series systems. COMS can support the processing of multiprogram transactions, single-station remote files, and multistation remote files.

compact data set

A collection of related data records stored in a file. These records are not maintained in logical order. The length of each record can vary because the data set items can vary in size or in number of occurrences, or because the data set items are stored conditionally.

COMS

See Communications Management System.

control file

A file containing data file coordination information, audit control information, and dynamic database parameter values.

controlpoint

A feature that limits the amount of audit information scanned by halt/load and Abort recovery. The controlpoint is performed when the last process leaves transaction state. Halt/load and Abort recovery do not process any audit images prior to the next to last controlpoint in the audit.

counted link

A link that has a designated count item in the data set to which the link points.

D

DASDL

See Data and Structure Definition Language.

Data and Structure Definition Language (DASDL)

The language used to describe a database logically and physically, and to specify criteria to ensure the integrity of data stored in the database. DASDL is the source language that is input to the DASDL compiler, which creates or updates the database description file from the input.

data independence
The property that establishes the ability to change the structural format of a database without requiring changes to or recompilation of unaffected application programs that use the database.

data item
A field in a database record or transaction format that contains a particular type of information.

Data Management System II (DMSII)
A specialized system software package used to describe a database and maintain the relationships among the data elements in the database.

data set
A collection of related data records stored in a file on a random-access storage device. A data set is similar to a conventional file. It contains data items and has logical and physical properties similar to files. However, unlike conventional files, data sets can contain other data sets, sets, and subsets.

database
An integrated, centralized system of data files and program utilities designed to support an application. The data sets and associated index structures are defined by a single description. Ideally, all the permanent data pertinent to a particular application resides in a single database. The database is considered a global entity that several applications can access and update concurrently.

deadlock
A situation in which two or more programs have locked records and are also attempting to lock records held by each other.

direct data set
A collection of related data records stored in a file. These records are maintained in key value order. One unsigned numeric data item in the record is designated as the key item.

discontinuity
The occurrence of a change in a structure that causes the structure to no longer match its audit image. Discontinuities affect audit and recovery.

disjoint
Pertaining to a data set, set, or subset when it is not contained in another data set. *Contrast with* embedded.

DMSII
See Data Management System II.

DMSUPPORT library
An object code file containing the procedures that are tailored for a particular database.

dump
A copy of a database on a tape.

E

embedded
Pertaining to a data set, set, or subset contained within another data set. A record of an embedded structure must be accessed through the *master* data set in which it is embedded. *Contrast with* disjoint.

end of file (EOF)
A code at the end of a data file that signals that the last record in the file has been processed.

EOF
See end of file.

ERGO
See Extended Retrieval with Graphic Output.

Extended Retrieval with Graphic Output (ERGO)
A program for Data Management System II (DMSII) database and file access, and for report generation.

F

FIB
See File Information Block.

field item
A data item that can contain unsigned integer or Boolean values.

File Information Block (FIB)
A data structure in an object code file that contains information describing a file.

G

garbage collection
The process of consolidating deleted or unused space in data sets, sets, and subsets, and returning this space to the system.

global data item
A data item, group item, or population item that is not a part of any data set. Global data items generally contain information such as control totals, hash totals, and populations that apply to the entire database.

group item
A collection of data items that can be viewed as a single data item.

guard file
A disk file created by the GUARDFILE utility program that describes the access rights of various users and programs to a program, data file, or database.

H

halt/load
A system-initialization procedure that temporarily halts the system and loads the operating system from a disk to main memory.

hashing algorithm
An algorithm used in constructing and maintaining hash tables, which modify record keys to produce the addresses of the record keys in a structure.

I

I/O
Input/output. An operation in which the system reads data from or writes data to a file on a peripheral device such as a disk drive.

index
A computer storage location, the contents of which identify a particular element in a table.

index random structure
A structure of records allocated to particular tables based on a hashing function of the key.

index sequential structure
A structure that is a collection of tables arranged hierarchically.

integer
A whole number.

interpretive interface
A set of conventions for passing information using the DMINTERPRETER library.

K

key item
A data item or group item that serves as a retrieval key for a set, subset, or access.

L

link
A field that enables one data set record to refer to another.

logical database
A collection of structures declared in the Data and Structure Definition Language (DASDL) that provide a view of the database, enforce structure-level security, and achieve data independence. When a logical database is declared in DASDL, the data sets, sets, subsets, and remaps to be included in it are listed.

M

manual subset
A subset that has no condition specifying which data set records are to be included in the subset. The user must add and delete manual subset entries, using the INSERT and REMOVE statements.

master
A data set or record that contains one or more embedded data sets or records. The terms master, parent, and owner are synonymous.

Master Control Program (MCP)
An operating system on A Series systems. The MCP controls the operational environment of the system by performing job selection, memory management, peripheral management, virtual memory management, dynamic subroutine linkage, and logging of errors and system utilization.

master data set
A data set that contains an embedded data set.

MCP
See Master Control Program.

MCS
See message control system.

message control system (MCS)
A program that controls the flow of messages between terminals, application programs, and the operating system. MCS functions can include message routing, access control, audit and recovery, system management, and message formatting.

N

numeric item
A data item whose description restricts its contents to a value represented by numeric characters.

O

OCM
See Operations Control Manager.

Operations Control Manager (OCM)
A menu-driven member of the InfoExec™ family of products that is used to maintain databases.

InfoExec is a trademark of Unisys Corporation.

ordered data set
A collection of related data records stored in a file in a physical sequence based on a user-specified key.

ordered list structure
A structure that allows retrieval of data set records based on a key value.

P

partitioned structure
A structure that can be declared and assigned one or more structure numbers corresponding to a data set, set, or subset.

physical database
An entire database as it is stored on a disk.

PL/I
Programming Language I. A high-level, structured programming language designed primarily for scientific and commercial use.

Q

quiet point
A time during which there are no transactions in progress for a database. *See also* syncpoint.

R

random data set
A collection of related data records stored in a file. These records are maintained in a physical sequence based on a hashing function of the record key.

RDS
See restart data set.

real item
A data item that stores signed or unsigned, fractional or whole values in single-precision, floating-point form.

rebuild
A recovery process in which the entire database is loaded from one or more sets of dump tapes. The recovery process then applies the audit trail after-update record images to move the database forward in time.

record
A group of logically related items of data in a file that are treated as a unit.

recovery

A procedure that is initiated following a hardware, software, or operations failure while the database is in update mode. Recovery backs out any partially completed transactions by applying audit-trail images to the database to restore it to a consistent state. In addition, recovery passes restart information to the programs accessing the database.

remap

A logical data set that redefines a physical data set by omitting, reordering, or renaming the items.

reorganization

The process of reordering or reformatting data sets, sets, or subsets. Reorganization can restore space in files, reorder data sets for more efficient retrieval, and reformat data set records when items are added, deleted, or changed.

restart data set (RDS)

A data set containing restart records that application programs can access to recover database information after a system failure.

rollback

The recovery of a database or transaction base to a consistent state at an earlier point in time.

RPG

Report Program Generator. A high-level, commercially oriented programming language used most frequently to produce reports based on information derived from data files.

S

set

A file of indexes that refers to all the records of a single data set. Sets are automatically maintained by the system. Sets permit access to the records of a data set in some logical sequence and are normally used to optimize certain types of retrievals of the data set records.

SMFII

See System Management Facility II.

structure

A data set, set, subset, access, or remap.

subset

An index structure that is identical to a set, except that the subset need not contain a record for every record of the data set. A set must index every record in its associated data set, whereas a subset can index zero, one, several, or all data set records. A subset might or might not be automatically maintained by DMSII.

syncpoint

A point in time when no program is in a transaction state.

System Management Facility II (SMFII)
A software system that monitors and provides data on four aspects of system performance: hardware, software, workload characterization, and system utilization.

T

table
A one-dimensional or multidimensional structure in which like data items are stored. Each data item can be uniquely identified and accessed by means of its location in the table; identification and access procedures vary according to the language or product.

timestamp
An encoded, 48-bit numerical value for the time and date. Various timestamps are maintained by the system for each disk file. Timestamps note the time and date a file was created, last altered, and last accessed.

TPS
See Transaction Processing System.

transaction
A sequence of operations grouped by a user program because the operations constitute a single logical change to the database.

Transaction Processing System (TPS)
A Unisys system that provides methods for processing a high volume of transactions, keeps track of all input transactions that access the database, enables the user to batch data for later processing, and enables transactions to be processed on a database that resides on a remote system.

transaction trail
A file maintained by a Communications Management System (COMS) database (DB) library that contains a series of time-ordered transactions that can be reapplied to the database to provide synchronized recovery in the event of a transaction-state abort, system crash, or rollback. The file can also be used to provide a journal of both query and update transactions for security auditing, accounting, and statistical reporting. Each DB library has its own transaction trail.

U

unordered data set
A collection of related data records stored in a file in which the records are either fixed format or variable format.

unordered list structure
A structure that allows retrieval of data set records in physical order.

update
To delete, insert, or modify information in a database or transaction base.

W

WFL

See Work Flow Language.

Work Flow Language (WFL)

A Unisys language used for constructing jobs that compile or run programs on A Series systems. WFL includes variables, expressions, and flow-of-control statements that offer the programmer a wide range of capabilities with regard to task control.

Bibliography

A Series ALGOL Programming Reference Manual, Volume 2: Product Interfaces (form 1170099). Unisys Corporation.

A Series CANDE Operations Reference Manual (form 1169869). Unisys Corporation.

A Series COBOL ANSI-74 Programming Reference Manual, Volume 2: Product Interfaces (form 1180619). Unisys Corporation.

A Series Communications Management System (COMS) Operations Guide (form 1154523). Unisys Corporation.

A Series DMSII Application Program Interfaces Programming Guide (form 5044225). Unisys Corporation. Formerly *A Series DMSII User Language Interface Programming Guide* .

A Series DMSII Data and Structure Definition Language (DASDL) Programming Reference Manual (form 1163805). Unisys Corporation.

A Series DMSII DataAid User's Guide (form 1180544). Unisys Corporation.

A Series DMSII Interpretive Interface Programming Reference Manual (form 1163813). Unisys Corporation.

A Series DMSII Transaction Processing System (TPS) Programming Guide (form 1164043). Unisys Corporation.

A Series DMSII Utilities Operations Guide (form 1163839). Unisys Corporation.

A Series Extended Retrieval with Graphic Output (ERGO) Operations Guide (form 1164027). Unisys Corporation.

A Series Generalized Message Control System (GEMCOS) Operations Guide (form 1185329). Unisys Corporation.

A Series InfoExec™ Administration Guide (form 1195070). Unisys Corporation.

A Series InfoExec Operations Guide (form 1195120). Unisys Corporation.

A Series Pascal Programming Reference Manual, Volume 2: Product Interfaces (form 1170107). Unisys Corporation.

A Series Report Program Generator (RPG) Programming Reference Manual, Volume 2: Product Interfaces (form 1170115). Unisys Corporation.

A Series Security Features Operations and Programming Guide (form 1195203). Unisys Corporation.

A Series System Commands Operations Reference Manual (form 1169612). Unisys Corporation.

A Series System Management Facility II (SMFII) Query Operations Guide (form 5015696). Unisys Corporation.

Data Analysis for Data Base Design. D.R. Howe. London: Edward Arnold Ltd, 1983.

DMSII Primer, The, Volume 1. Sonoma: Gregory Publishing Company, 1982.

Index

A

abort, 6–8
  recovery, 6–8
ABORTTRANSACTION language extension, 6–9
Access, 2–7
  in ordered data sets, 2–9
  in random data sets, 2–8
  with MODULUS physical option, 2–36
accessing databases, 7–1
  interactions, 1–6
accessing structures
  optimization, 2–37
Accessroutines, 1–3
  access, 1–7
  audit trail copy, 6–4
  buffers, 6–7
  DMSUPPORT library, 1–4
  guard files, 3–7
  halt/load, 6–9
  reblocking, 2–37
ADDRESSCHECK physical option, 2–35, 2–36, 2–37
ADDS, (See Advanced Data Dictionary System (ADDS))
Advanced Data Dictionary System (ADDS), 4–16
afterimage, 6–4
aggregate items
  global data items, 2–13
  in embedded data sets, 2–34
ALLOCATE physical option, 2–35
ALLOWEDCORE parameter, 2–38
alpha items, 2–12
ALTERNATE audit trail attribute, 6–4
analyzing information for databases, 4–1
application program interfaces, 7–3
AREAS physical option, 2–35
AREASIZE physical option, 2–35
ASCENDING option for keys, 2–17
ASSIGN statement in transaction state, 6–4
attributes, audit trails, 6–4
audit files, 6–2
  consistency checks, 6–5
  copying and examining, 6–7
  discontinuity, 6–12
  location, 6–4
  row reconstruction, 6–11
  VISIBLE DBS, 8–3
AUDIT option, 6–2
audit trails, 6–2
  ALTERNATE attribute, 6–4
  attributes, 6–4
  changes to databases, 6–4
  DUPLICATED attribute, 6–4
  duplicates, 6–4
  KIND attribute, 6–4
  rebuilding with, 6–12
  rolling back with, 6–12
  UPDATE EOF attribute, 6–4
  VERIFY attribute, 6–5
audited databases, 6–2
auditing databases, 6–2
automatic recovery, 6–8
automatic subsets, 2–20
  advantages in creating databases, 4–5, 4–6
AVERAGE RECORD SIZE physical option, 2–36

B

backing out, 6–5
backing out transactions, 6–4
backing up databases, 6–1
beforeimage, 6–4
BEGINTRANSACTION statement, 6–3
bit vector sets and subsets
  disjoint restriction on, 2–34
bit vectors, 2–24
BLOCKSIZE physical option, 2–36

Boolean items, 2–12
bottom-up approach, designing databases, 4–3
BUFFERS physical option, 2–36
BUILDDUMPDIRECTORY statement, 6–2

C

certifying database integrity, 8–2
changes
  backing out, 6–4
  reapplying, 6–4
changing
  databases, 5–1
  keys, 2–16
CHECKSUM physical option, 2–36
choosing relationships, 4–7
collecting information for databases, 4–1
Communications Management System (COMS)
  synchronized recovery, 6–10
compact data sets, 2–7
compiling tailored software, 4–16
COMS, (See Communications Management System (COMS))
concepts related to DMSII databases, 2–1
concurrency control, 6–9
  INDEPENDENTTRANS, 6–9
  REAPPLYCOMPLETED, 6–9
connecting data set records, 2–25
control file, 1–4
  dump tape directory pack, 6–2
  initializing, 4–17
  offline dumps, 6–1
CONTROLPOINT parameter, 2–39, 6–7
controlpoints, 6–7
COPYAUDIT, 6–7
copying audit files, 6–7
count items, 2–13
count items, counted links, 2–28
counted links, 2–28
creating databases, 4–1
  choosing relationships, 4–7
  collecting and analyzing information, 4–1
  compiling tailored software, 4–16
  defining a database to the system, 4–15
  implementing, 4–4
  interactions, 1–5
  selecting an approach, 4–2

D

DASDL, (See Data and Structure Definition Language (DASDL))
data
  analyzing, 4–1
  collecting, 4–1
Data and Structure Definition Language (DASDL), 1–3
  compiler, 1–3
  source, writing, 4–16
data independence, 3–1
data items, 2–11
  alpha items, 2–12
  Boolean items, 2–12
  counting with aggregate items, 2–13
  data sets, 2–4
  defaults, (See physical options)
  field items, 2–12
  filler items, 2–13
  foreign key technique use of, 4–5
  global, 2–13
  global data item use, 2–13
  global items, 2–13
  group items, 2–12
  key use of, 2–15
  numeric items, 2–12
  options, 2–14
    INITIALVALUE option, 2–14
    NULL option, 2–14
    OCCURS option, 2–14
    SIZE option, 2–14
    STORED option, 2–15
    VERIFY option, 2–15
  real items, 2–12
  types (table), 2–11
Data Management System II (DMSII), 1–1
  framework, 1–2
data set defaults, 2–38
data set physical options, 2–38
data sets, 2–3
  ALLOCATE physical option, 2–35
  AREAS physical option, 2–35
  AREASIZE physical option, 2–35
  AVERAGE RECORD SIZE physical option, 2–36
  BLOCKSIZE physical option, 2–36
  BUFFERS physical option, 2–36
  compact, 2–7
  defaults, 2–35, (See also physical options)
    CHECKSUM physical option, 2–36
    DIGITCHECK physical option, 2–36

direct, 2–7
restrictions on disjoint, 2–34
disjoint, 2–33
advantages of, 4–4
default, 2–34
default type for, 2–34
DISK/PACK physical option, 2–36
embedded
using aggregate items with, 2–34
global defaults, 2–37
indexing records, 2–15
information, 2–3
links in, 2–25
LOCK TO MODIFY DETAILS physical option, 2–36
many-to-many (M:N) relationships
between two data sets, 4–14
within one data set, 4–12
many-to-many (M:N) Relationships
within one data set, 4–11
master, 2–34
MEMORY RESIDENT physical option, 2–36
one-to-many (1:M) relationships
between two data sets, 4–10
within one data set on one data set, 4–10
one-to-one (1:1) relationships
on one data set, 4–8
OPEN PARTITIONS physical option, 2–37
ordered, 2–9
embedded, 2–34
PARTITIONS physical option, 2–37
physical options, 2–35
POPULATION physical option, 2–37
random, 2–8
REBLOCK physical option, 2–37
REBLOCKFACTOR physical option, 2–37
relationship technique, 4–12
relationships
one data set, 4–12
two data sets, 4–14
restart, 6–7
SECURITYGUARD physical option, 2–37
standard, 2–5
SUBBLOCKSIZE physical option, 2–37
TABLESIZE physical option, 2–37
tabular information, 2–3
types, 2–20
types of, 2–4
unordered, 2–10
data sets, two
one-to-one (1:1) relationships
between two data sets, 4–8
database design
bottom-up approach, 4–3
top-down approach, 4–4
< database name > /CONTROL, 1–4
< database name > /DUMPDIR, 6–2
database recovery, 6–12
rebuilding, 6–12
rollback, 6–12
DATABASE/DMSUPPORT file, 1–4
DATABASE/PROPERTIES file, 2–37
databases
access, 7–1
accessing, interactions in, 1–6
approach to designing, 4–2
audit, 6–2
backup, 6–1
changing, 5–1
concepts, 2–1
creating, 4–1
DBANALYZER reports, 8–2
defaults, (See physical options)
defining to the system, 4–15
designing, 4–2
DMUTILITY program, 8–3
dumping, 6–1
global defaults, 2–37
initializing, 4–17
integrity, 8–2
interactions in, 1–5
interlock control, 1–4
logical, 3–3
maintaining, 8–1
modeling, 5–2
OCM tasks, 8–1
organizing, 4–3
parameters, 2–38
partitioned, 2–37
parts, 2–1
physical, 3–1
physical options, 2–35
recovery, 6–8
reorganizing, 5–1
restoring, 6–1
simplicity, 4–4
structures, 2–1, 2–2
unaudited, 6–2
updating, 5–1
VISIBLE DBS, 8–3
DBANALYZER program, 8–2
DBCERTIFICATION program, 8–2

Accessroutines, 1–4
defaults
global, 2–37
defining databases to the system, 4–15
DELETE statement in transaction state, 6–4
DESCENDING option for keys, 2–17
description file, 1–3
DMALGOL compiler, 4–17
SYSTEM/DASDL, 1–3
DESCRIPTION/< database name > file, 1–3
designing databases
bottom-up approach, 4–3
top-down approach, 4–4
designing for simplicity, 4–4
DIGITCHECK physical option, 2–36
direct data sets, 2–7
restrictions on disjoint, 2–34
discontinuity, 6–12
disjoint
data sets
advantages of, 4–4
default type for, 2–34
structures, 2–33, 2–35
table representation of, 2–33, 4–5
turning embedded structures into, 4–6
DISK/PACK physical option, 2–36
DMALGOL compiler and the description file, 4–17
DMDATARECOVERY/< database name > program, 6–11
DMINTERPRETER, 7–3
DMSUPPORT library, 1–4
DMMONITOR program, 8–3
DMRECOVERY program, 6–12
DMSAVEPOINT language extension, 6–9
DMSII, (See Data Management System II (DMSII))
DMSUPPORT library, 1–4
DMSUPPORT/< database name > file, 1–4
DMUTILITY program, 8–3
dumping, 6–1
initializing database files, 4–17
manual recovery, 6–11
databases, 6–11
dump tape directories
maintaining, 6–2
dumping
offline, 6–1
online, 6–1
DUPLICATED audit trail attribute, 6–4
duplicates
order of, not represented in automatic structures, 4–5
DUPLICATES option for keys, 2–16

E

embedded
data sets
disadvantages of, 2–33
hierarchy representation of, 4–5
using aggregate items with, 2–34
structures, 2–33, 2–35
turning into disjoint structures, 4–6
ENDTRANSACTION statement, 6–3
environment, DMSII, functions of, 1–1
ERGO, (See Extended Retrieval with Graphic Output (ERGO))
examining audit files, 6–7
Extended Retrieval with Graphic Output (ERGO), 7–3

F

FALSE values, Boolean items, 2–12
field items, 2–12
files, format conversion, 5–1
filler items, 2–13
FIRST, DUPLICATES, 2–16
fixed-format standard data sets, 2–5
foreign key technique, 4–5
format conversion, 5–1
framework of DMSII, 1–2

G

garbage collection, 5–2
GENERATE statement in transaction state, 6–4
global
data items, 2–13
aggregate items, 2–13
population, 2–13
defaults, 2–37, (See also physical options)
BUFFERS physical option, 2–36
CHECKSUM physical option, 2–36
DISK/PACK physical option, 2–36
LOCK TO MODIFY DETAILS physical option, 2–36

REBLOCK physical option, 2–37
REBLOCKFACTOR physical option, 2–37
SECURITYGUARD physical option, 2–37
group items, 2–12
guard files, 3–6
GUARDFILE option, security, 3–7
GUARDFILE utility, 3–6

H

halt/load recovery, 6–8
hashing algorithm
in index sequential structures, 2–21
in random data sets, 2–8
HIDDEN option for remaps, 3–3
hierarchy
embedded structures
representation of, 4–5
HIGH-VALUE, INITIALVALUE option, 2–14

I

INDEPENDENTTRANS option, 6–9
syncpoints, 6–6
index random sets and subsets, 2–21
LOADFACTOR physical option, 2–36
MODULUS physical option, 2–36
restrictions on disjoint, 2–34
index sequential sets and subsets, 2–20
advantages of, 4–6
LOADFACTOR physical option, 2–36
indexing data set records, 2–15
information
analyzing, 4–1
collecting, 4–1
information in data sets, 2–3
INITIALIZE statement, 4–17
initializing databases, 4–17
INITIALVALUE data item option, 2–14
INSERT statement in transaction state, 6–4
integrity of the database, certifying, 8–2
interactions in
accessing databases, 1–6
creating databases, 1–5
interfaces, program, 7–3
interlock, controlling databases, 1–4
interpretive interfaces, 7–3

K

KEYCHANGEOK option of NO DUPLICATES option, 2–16
keys, 2–15
changing, 2–16
data items as, 2–15
foreign technique, 4–5
in an Access, 2–7
in direct data sets, 2–7
in ordered data sets, 2–9
in random data sets, 2–8
in sets, 2–15
in subsets, 2–15
master record, 2–34
ordering entries with, 2–16
restricting set and subset membership with, 2–16
KIND audit trail attribute, 6–4

L

language extensions, 7–3
LAST, DUPLICATES, 2–16
levels, update, 5–1
links, 2–25
counted, 2–28
disadvantages of, 2–26
foreign key technique to substitute for, 4–5
index sequential sets, comparison, 4–6
self-correcting, 2–31
symbolic, 2–29
type of, 2–26
unprotected, 2–27
verified, 2–27
LOADDUMP program, 8–3
LOADFACTOR physical option, 2–36
LOCK TO MODIFY DETAILS physical option, 2–36
logical databases, 3–3
LOW-VALUE, INITIALVALUE option, 2–14

M

main directory, 6–2
maintaining

databases, 8–1
dump tape directories, 6–2
manual recovery, 6–11
database, 6–12
partial, 6–11
manual subsets, 2–20
automatic subsets, comparison, 4–6
many-to-many (M:N) relationships, 4–11
Master Control Program (MCP) and the Accessroutines, 1–4
master data sets for embedded structures, 2–34
master record in keys, 2–34
master records
locking with LOCK TO MODIFY DETAILS physical option, 2–36
MEMORY RESIDENT physical option, 2–36
MIDTRANSACTION statement, 6–10
modeling databases, 5–2
MODULUS option
for random data sets, 2–8
MODULUS physical option, 2–36

N

NO DUPLICATES option for keys, 2–16
NOZIP option, 6–12
NULL data item option, 2–14
numeric items, 2–12

O

OCCURS data item option, 2–14
OCM, (See Operations Control Manager (OCM))
offline dumps, 6–1
one-to-many (1:M) relationships, 4–9
one-to-one (1:1) relationships, 4–7
online dumps, 6–1
OPEN PARTITIONS physical option, 2–37
Operations Control Manager (OCM), 8–1
options, (See data items)
data items, 2–14
physical, 2–35
ordered data sets, 2–9
usually embedded, 2–34
ordered list sets and subsets
default, 2–34
LOADFACTOR physical option, 2–36
ordered lists, 2–22
ordering data set records, 2–15
organization of the database, 4–3
OVERLAYGOAL parameter, 2–38

P

parameters, databases, 2–38
partial database recovery, 6–11
quickfix, 6–11
row reconstruction, 6–11
partitioned structures
data set physical option, 2–37
PARTITIONS physical option, 2–37
parts of DMSII databases, 2–1
physical databases, 3–1
physical options, 2–35
ALLOCATE option, 2–35
AREAS option, 2–35
AREASIZE option, 2–35
population global data items, 2–13
POPULATION physical option, 2–37
PRIME physical option, 2–37
PRINTAUDIT, 6–7
programs
application, 7–3
interfaces, 7–3

Q

quickfix process, 6–11
quiet points, 6–5

R

random access, optimization, 2–37
random data sets, 2–8
READONLY ALL option for remaps, 3–3
real items, 2–12
REAPPLYCOMPLETED option, 6–9
creating syncpoints, 6–6
reapplying transactions, 6–4
REBLOCK physical option, 2–37
REBLOCKFACTOR physical option, 2–37
rebuilding databases, 6–12
RECONSTRUCT program, 6–11
RECONSTRUCT/< database name >, 6–11
reconstruction, row, 6–11

record-type items, 2–6
recording transactions, 6–3
records, 2–4, (See also relationships)
  beforeimage, 6–4
  bits, represented by, 2–24
  compacting, 2–7
  compacting with STORED data item option, 2–15
  connecting, 2–25
  counting with POPULATION global data item, 2–13
  fixed-format, 2–5
  format conversion, 5–1
  indexing in data sets, 2–15
  links for connections, 2–25
  master, 2–34
  number of with MODULUS option, 2–8
  ordering with Access, 2–7
  simple, 4–4
  size with BLOCKSIZE physical option, 2–36
  space
    allocating, 2–35
    reserving, 2–13
  transaction changes to, 6–3
  variable-format, 2–6
RECOVER statement, 6–12
recovery, 6–8
  abort, 6–8
  automatic, 6–8
  halt/load, 6–8
  manual, 6–11
  partial database, 6–11
  quickfix, 6–11
  rebuild, 6–12
  rollback, 6–12
  row, 6–11
  row reconstruction, 6–11
  synchronized, 6–10
  whole database, 6–12
relationship data set between two data sets, 4–14
relationship data set on one data set, 4–12
relationship data set technique, 4–12
relationships, 4–7
  many-to-many (M:N)
    between two data sets, 4–14
    within one data set, 4–12
  one-to-many (1:M)
    between two data sets, 4–10
    within one data set, 4–10
  one-to-one (1:1)
    between two data sets, 4–8
    within one data set, 4–8
remaps, 3–1
REMOVE statement in transaction state, 6–4
reorganizing databases, 5–1
  format conversion, 5–1
  garbage collection, 5–2
reports
  DBANALYZER program, 8–2
  ERGO, 7–3
REQUIRED ALL option for remaps, 3–3
REQUIRED data item option, 2–14
restart data sets, 6–7
restoring databases, 6–1
rollback process, 6–12
row reconstruction, 6–11
row recovery, 6–11

S

security, 3–1
  with guard files, 3–6
  with logical databases, 3–3
  with remaps, 3–1
SECURITYGUARD physical option, 2–37
  security, 3–7
SELECT condition, remaps, 3–3
self-correcting links, 2–31
serial access, optimization, 2–37
sets, 2–17, 2–35
  AREAS physical option, 2–35
  AREASIZE physical option, 2–35
  automatic and manual comparison, 4–6
  bit vectors, 2–24
    disjoint, 2–34
  defaults, 2–38, (See also physical options)
  DISK/PACK physical option, 2–36
  embedded
    default, 2–34
  global default, 2–37
  index random, 2–21
    disjoint, 2–34
  index sequential, 2–20
    compared with links, 4–6
    creating databases, 4–6
  keys, 2–16
  MEMORY RESIDENT physical option, 2–36
  MODULUS physical option, 2–36
  ordered lists, 2–22
    default, 2–34

physical options, 2–35, 2–38
PRIME physical option, 2–37
TABLESIZE physical option, 2–37
types, 2–20
unordered lists, 2–23
simple records, 4–4
simple structures, 4–4
simplifying databases, 4–4
SIZE data item option, 2–14
source, Data and Structure Definition Language (DASDL), 1–3
space
compacting with STORED option, 2–15
records
saving, 2–7
reserving for future information, 2–13
standard data sets, 2–5
fixed format, 2–5
variable format, 2–6
statistics
DMMONITOR, 8–3
STORE statement in transaction state, 6–4
STORED data item option, 2–15
structures
access optimization to, 2–37
buffer allocation for, 2–36
data set definition, 2–3
databases, 2–1, 2–2
DBANALYZER program, 8–2
discontinuity, 6–12
disjoint, 2–33
DMUTILITY program, 8–3
embedded, 2–33
to disjoint, 4–6
global data items, 2–13
global default, 2–37
logical, 3–1
logical organization of databases, 2–35
modeling, 5–2
partitioned, 2–37
restricting access, 3–3
restrictions, 2–34
simple, 4–4
tasks with OCM, 8–1
verification during dumping, 6–1
VISIBLE DBS, 8–3
SUBBLOCKSIZE physical option, 2–37
subsets, 2–19, 2–35
AREAS physical option, 2–35
AREASIZE physical option, 2–35
automatic, 2–20
creating databases, 4–6
automatic and manual comparison, 4–6
bit vectors, 2–24
defaults, 2–38, (See also physical options)
DISK/PACK physical option, 2–36
embedded
default, 2–34
global default, 2–37
index random, 2–21
disjoint, 2–34
index sequential, 2–20
compared with links, 4–6
keys, 2–16
manual, 2–20
MEMORY RESIDENT physical option, 2–36
MODULUS physical option, 2–36
ordered lists, 2–22
default, 2–34
physical options, 2–35, 2–38
PRIME physical option, 2–37
TABLESIZE physical option, 2–37
types, 2–20
unordered lists, 2–23
symbolic links, 2–29
synchronized recovery, COMS, 6–10
SYNCPOINT parameter, 2–39
syncpoints, 6–6
SYNCWAIT parameter, 6–6
SYSTEM/ACCESSROUTINES, 1–3, (See also Accessroutines)
SYSTEM/COPYAUDIT, 6–7
SYSTEM/DASDL, 1–3
SYSTEM/DMCONTROL, 1–4
control file initialization, 4–17
SYSTEM/DMINTERFACE program, 7–3
SYSTEM/DMUTILITY, (See DMUTILITY program)
SYSTEM/GUARDFILE utility, 3–6
SYSTEM/PRINTAUDIT, 6–7

T

tables
allocated by MODULUS physical option, 2–36
description file use of, 1–3
disjoint structures representation of, 4–5
disjoint structures represented as, 2–33
index sequential structures use of, 2–20, 2–21

MODULUS option declares the number of, 2–8
TABLESIZE physical option, 2–37
TABLESIZE physical option, 2–37
tailored software, 4–16
top-down approach, designing databases, 4–4
transaction
state, 6–3
trail, 6–10
transactions, 6–3
backing out, 6–4
reapplying, 6–4
recording, 6–3
recovery, 6–5
restart data sets, 6–7
TRUE values, Boolean items, 2–12
types of data items, 2–11
types of data sets, 2–4
types of links, 2–26
types of sets, 2–20

U

unaudited databases, 6–2
unordered data sets, 2–10
unordered lists, 2–23
unprotected links, 2–27
UPDATE EOF audit trail attribute, 6–4
update levels, 5–1
updates, 6–1
updating databases, 5–1

V

variable-format standard data sets, 2–6
verified links, 2–27
VERIFY
data item option, 2–15
VERIFY audit trail attribute, 6–5
VISIBLE DBS, 8–3

W

WHERE clause, 2–20
whole database recovery, 6–12
recovery, 6–12
rollback, 6–12

UNISYS

Help Us To Help You

Publication Title

Form Number

Unisys Corporation is interested in your comments and suggestions reguarding this manual. We will use them to improve the quality of your Product Information. Please check type of suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments:

Name Telephone number ( )

Title Company

Address

City State Zip code

Cut along dotted line

Tape

Please Do Not Staple

Tape

Fold Here

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 817 DETROIT, MI

POSTAGE WILL BE PAID BY ADDRESSEE

UNISYS CORPORATION
ATTN: PUBLICATIONS
25725 JERONIMO ROAD
MISSION VIEJO, CA 92691-9826

5044191000000380